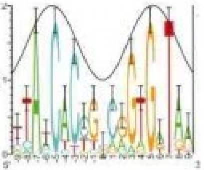
*Wiley Series in Bioinformatics • Yi Pan and Albert Y. Zomaya, Series Editors*



# ALGORITHMS IN COMPUTATIONAL MOLECULAR BIOLOGY

## Techniques, Approaches and Applications

EDITED BY

MOURAD ELLOUMI

ALBERT Y. ZOMAYA

WILEY

# ALGORITHMS IN COMPUTATIONAL MOLECULAR BIOLOGY

Wiley Series on

**Bioinformatics: Computational Techniques and Engineering**

A complete list of the titles in this series appears at the end of this volume.

# ALGORITHMS IN COMPUTATIONAL MOLECULAR BIOLOGY

## Techniques, Approaches and Applications

Edited by

Mourad Elloumi
Unit of Technologies of Information and Communication
and University of Tunis-El Manar, Tunisia

Albert Y. Zomaya
The University of Sydney, Australia

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada.

Limit of Liability/Disclaimer of Warranty: While the publisher and the author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor the author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information about our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

***Library of Congress Cataloging-in-Publication Data is available.***

ISBN: 978-0-470-50519-9

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To our families, for their patience and support.

# CONTENTS

# PREFACE

Computational molecular biology has emerged from the *Human Genome Project* as an important discipline for academic research and industrial application. The exponential growth of the size of biological databases, the complexity of biological problems, and the necessity to deal with errors in biological sequences require the development of fast, low-memory requirement and high-performance algorithms. This book is a forum of such algorithms, based on new/improved approaches and/or techniques. Most of the current books on algorithms in computational molecular biology either lack technical depth or focus on specific narrow topics. This book is the first overview on algorithms in computational molecular biology with both a wide coverage of this field and enough depth to be of practical use to working professionals. It surveys the most recent developments, offering enough fundamental and technical information on these algorithms and the related problems without overloading the reader. So, this book endeavors to strike a balance between theoretical and practical coverage of a wide range of issues in computational molecular biology. Of course, the list of topics that is explored in this book is not exhaustive, but it is hoped that the topics covered will get the reader to think of the implications of the presented algorithms on the developments in his/her own field. The material included in this book was carefully chosen for quality and relevance. This book also presents a mixture of experiments and simulations that provide not only qualitative but also quantitative insights into the rich field of computational molecular biology. It is hoped that this book will increase the interest of the algorithmics community in studying a wider range of combinatorial problems that originate in computational molecular biology. This should enable researchers to deal with more complex issues and richer data sets.

Ideally, the reader of this book should be someone who is familiar with computational molecular biology and would like to learn more about algorithms that deal with the most studied, the most important, and/or the newest topics in the field of computational molecular biology. However, this book could be used by a wider audience such as graduate students, senior undergraduate students, researchers, instructors, and practitioners in computer science, life science, and mathematics. We have tried to make the material of this book self-contained so that the reader would not have to consult a lot of external references. Thus, the reader of this book will certainly find what he/she is looking for or at least a clue that will help to make an advance in his/her research. This book is quite timely, because the field of computational molecular biology as a whole is undergoing many changes, and will be of a great use to the reader.

This book is organized into seven parts: *Strings Processing and Application to Biological Sequences, Analysis of Biological Sequences, Motif Finding and Structure Prediction, Phylogeny Reconstruction, Microarray Data Analysis, Analysis of Genomes, and Analysis of Biological Networks.* The 42 chapters, that make up the seven parts of this book, were carefully selected to provide a wide scope with minimal overlap between the chapters in order to reduce duplication. Each contributor was asked that his/her chapter should cover review material as well as current developments. In addition, we selected authors who are leaders in their respective fields.

MOURAD ELLOUMI AND ALBERT Y. ZOMAYA

# CONTRIBUTORS

**Bassam A. Alqaralleh,** Faculty of IT, Al-Hussien Bin Talal University, Jordan.

**Srinivas Aluru,** Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA; and Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India.

**Mohamed Radhouene Aniba,** Institute of Genetics and Molecular and Cellular Biology, Illkirch, France.

**Pavlos Antoniou,** Department of Computer Science, King's College, London, UK.

**Wassim Ayadi,** Unit of Technologies of Information and Communication (UTIC) and University of Tunis-El Manar, Tunisia.

**Enrique Blanco,** Department of Genetics, Institute of Biomedicine of the University of Barcelona, Spain.

**Guillaume Blin,** IGM, University Paris-Est, Champs-sur-Marne, Marne-la-Vallée, France.

**Dragan Bosnacki,** Eindhoven University of Technology, The Netherlands.

**Jérémie Bourdon,** LINA, University of Nantes and INRIA Rennes-Bretagne-Atlantique, France.

**Carlo Cattani,** Department of Mathematics, University of Salerno, Italy.

**Elsa Chacko,** Department of Chemistry and Biomolecular Sciences and ARC Centre of Excellence in Bioinformatics, Macquarie University, Sydney, Australia.

**Raymond H. F. Chan,** Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong, China.

**Luonan Chen,** Key Laboratory of Systems Biology, Shanghai Institutes for Biological Sciences, Chinese Academy of Sciences, Shanghai, China.

**Hsin-Hung Chou,** Department of Information Management, Chang Jung Christian University, Tainan, Taiwan.

**Manolis Christodoulakis,** Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus; and Department of Computer Science, King's College London, London, UK.

**Adrian Cootes,** Macquarie University, Sydney, Australia.

**Maxime Crochemore,** IGM, University Paris-Est, Champs-sur-Marne, Marne-la-Vallée, France.

**Bhaskar DasGupta,** Department of Computer Science, University of Illinois at Chicago, USA.

**Amitava Datta,** School of Computer Science and Software Engineering, The University of Western Australia, Perth, Australia.

**Erik P. de Vink,** Eindhoven University of Technology, The Netherlands.

**Wei Du,** College of Computer Science and Technology, Jilin University, Changchun, China.

**Mohamed Elati,** Institute of Systems and Synthetic Biology, Evry University - Genopole, Evry, France.

**Mourad Elloumi,** Unit of Technologies of Information and Communication (UTIC) and University of Tunis-El Manar, Tunisia.

**Chiara Epifanio,** Department of Mathematics and Applications, University of Palermo, Italy.

**Patricia A. Evans,** Faculty of Computer Science, University of New Brunswick, Fredericton, Canada.

**Damien Eveillard,** LINA, University of Nantes and INRIA Rennes-Bretagne-Atlantique, France.

**Tarek El Falah,** Unit of Technologies of Information and Communication (UTIC) and University of Tunis-El Manar, Tunisia.

**Guillaume Fertin,** LINA UMR CNRS 6241, University of Nantes, France.

**Alessandra Gabriele,** Department of Mathematics and Applications, University of Palermo, Italy.

**Jennifer Gamble,** Vascular Biology Laboratory, Centenary Institute, Sydney, Australia.

**Xiangchao Gan,** The Wellcome Trust Center for Human Genetics, University of Oxford, UK.

**Raffaele Giancarlo,** Department of Mathematics and Applications, University of Palermo, Italy.

**Mathieu Giraud,** LIFL, University of Lille 1 and INRIA Lille - Nord Europe, Villeneuve d'Ascq, France.

**Adrien Goëffon,** LERIA, University of Angers, France.

**Jin-Kao Hao,** LERIA, University of Angers, France.

**Masud Hasan,** Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh.

**Peter A. J. Hilbers,** Eindhoven University of Technology, The Netherlands.

**Jan Holub,** Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic.

**Sun-Yuan Hsieh,** Department of Computer Science and Information Engineering, Institute of Medical Informatics, Institute of Manufacturing Information and Systems, National Cheng Kung University, Tainan, Taiwan.

**Chao-Wen Huang,** Department of Computer Science and Information Engineering, National Cheng Kung University Tainan, Taiwan.

**Costas S. Iliopoulos,** Department of Computer Science, King's College London, London, UK & Digital Ecosystems & Business Intelligence Institute, Curtin University, Perth, Australia.

**Ming-Yang Kao,** Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA

**Radha Krishna Murthy Karuturi,** Computational and Systems Biology, Genome Institute of Singapore.

**Varun Khanna,** Department of Chemistry and Biomolecular Sciences, and ARC Centre of Excellence in Bioinformatics, Macquarie University Sydney, Australia.

**Gaurav Kumar,** Department of Chemistry and Biomolecular Sciences, Macquarie University, Sydney, Australia

**Vamsi Kundeti,** Department of Computer Science and Engineering, University of Connecticut, Storrs, USA.

**Thierry Lecroq,** LITIS, University of Rouen, France.

**Yanchun Liang,** College of Computer Science and Technology, Jilin University, Changchun, China.

**Jana Sperschneider,** School of Computer Science and Software Engineering, The University of Western Australia, Perth, Australia.

**Alan Wee-Chung Liew,** School of Information and Communication Technology, Griffith University, Australia.

**Christos Makris,** Computer Engineering and Informatics Department, University of Patras, Rio, Greece.

**Ion Mandoiu,** Computer Science & Engineering Department, University of Connecticut, Storrs, CT, USA.

**Ronny S. Mans,** Eindhoven University of Technology, The Netherlands.

**Ahmed Mokaddem,** Unit of Technologies of Information and Communication (UTIC) and University of Tunis-El Manar, Tunisia.

**Giulio Pavesi,** Department of Biomolecular Sciences and Biotechnology, University of Milan, Italy.

**Pierre Peterlongo,** INRIA Rennes Bretagne Atlantique, Campus de Beaulieu, Rennes, France.

**Nadia Pisanti,** Dipartimento di Informatica, University of Pisa, Italy.

**Yu-Qing Qiu,** Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China.

**Mohammed S. Rahman,** Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh.

**Sanguthevar Rajasekaran,** Department of Computer Science and Engineering, University of Connecticut, Storrs, USA.

**Shoba Ranganathan,** Department of Chemistry and Biomolecular Sciences, and ARC Centre of Excellence in Bioinformatics, Macquarie University Sydney, Australia and Department of Biochemistry, Yong Loo Lin School of Medicine, National University of Singapore, Singapore.

**Jean-Michel Richer,** LERIA, University of Angers, France.

**Eric Rivals,** LIRMM, University Montpellier 2, France.

**Céline Rouveirol,** LIPN, UMR CNRS, Institute Galilée, University Paris-Nord, France.

**Irena Rusu,** LINA UMR CNRS 6241, University of Nantes, France.

**Leena Salmela,** Department of Computer Science, University of Helsinki, Finland.

**Martin Schiller,** School of Life Sciences, University of Nevada Las Vegas, USA.

**Marinella Sciortino,** Department of Mathematics and Applications, University of Palermo, Italy.

**Eduardo Sontag,** Department of Mathematics, Rutgers, The State University of New Jersey, Piscataway, NJ, USA.

**Jana Sperschneider,** School of Computer Science and Software Engineering, The University of Western Australia, Perth, Australia.

**Alexandros Stamatakis,** The Exelixis Lab, Department of Computer Science, Technische Universität München, Germany.

**Jorma Tarhio,** Department of Computer Science and Engineering, Aalto University, Espoo, Finland.

**Evangelos Theodoridis,** Computer Engineering and Informatics Department, University of Patras, Rio, Greece.

**Julie Thompson,** Institute of Genetics and Molecular and Cellular Biology, Illkirch, France.

**Mathew Vadas,** Vascular Biology Laboratory, Centenary Institute, Sydney, Australia.

**Paola Vera-Licona,** Institut Curie and INSERM, Paris, France.

**Stéphane Vialette,** IGM, University Paris-Est, Champs-sur-Marne, Marne-la-Vallée, France.

**Chen Wang,** CSIRO ICT Centre, Australia.

**Roger W. Wang,** Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong, China.

**Shuqin Wang,** College of Computer Science and Technology, Jilin University, Changchun, China.

**Yan Wang,** College of Computer Science and Technology, Jilin University, Changchun, China.

**H. Todd Wareham,** Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada.

**Jeff C. F. Wong,** Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong, China.

**Ling-Yun Wu,** Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China.

**Xiao Yang,** Department of Electrical and Computer Engineering, Bioinformatics and Computational Biology program, Iowa State University, Ames, IA, USA.

**Paul D. Yoo,** School of information Technologies, The University of Sydney, Australia.

**Federico Zambelli,** Department of Biomolecular Sciences and Biotechnology, University of Milan, Italy.

**Chen Zhang,** College of Computer Science and Technology, Jilin University, Changchun, China.

**Shihua Zhang,** Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China.

**Xiang-Sun Zhang,** Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China.

**Xing-Ming Zhao,** Institute of Systems Biology, Shanghai University, China.

**Bing Bing Zhou,** School of information Technologies, The University of Sydney, Australia.

**Chunguang Zhou,** College of Computer Science and Technology, Jilin University, Changchun, China.

**You Zhou,** College of Computer Science and Technology, Jilin University, Changchun, China.

**Albert Y. Zomaya,** School of Information Technologies, The University of Sydney, Australia.

# I

# STRINGS PROCESSING AND APPLICATION TO BIOLOGICAL SEQUENCES

# 1

# STRING DATA STRUCTURES FOR COMPUTATIONAL MOLECULAR BIOLOGY

Christos Makris and Evangelos Theodoridis

## 1.1 INTRODUCTION

The topic of the chapter is string data structures with applications in the field of computational molecular biology. Let $\Sigma$ be a finite alphabet consisting of a set of characters (or symbols). The cardinality of the alphabet denoted by $|\Sigma|$ expresses the number of distinct characters in the alphabet. A *string* or *word* is an ordered list of zero or more characters drawn from the alphabet. A word or string $w$ of length $n$ is represented by $w[1 \cdots n] = w[1]w[2] \cdots, w[n]$, where $w[i] \in \Sigma$ for $1 \leq i \leq n$ and $|w|$ denotes the length of $w$. The empty word is the empty sequence (of zero length) and is denoted by $\varepsilon$. A list of characters of $w$, appearing in consecutive positions, is called a substring of $w$, denoted by $w[i \cdots j]$, where $i$ and $j$ are the starting and ending positions, respectively. If the substring starts at position 1, then it is called a prefix, whereas if it ends at position $n$, then it is called a suffix of $w$. However, an ordered list of characters of $w$ that are not necessarily consecutive is called a *subsequence* of $w$.

Strings and subsequences appear in a plethora of computational molecular biology problems because the basic types of DNA, RNA, and protein molecules can be represented as strings—pieces of DNA as strings over the alphabet $\{A, C, G, T\}$ (representing the four bases adenine, cytosine, guanine, and thymine, respectively), pieces of RNA as strings over the alphabet $\{A, C, G, U\}$ (with uracil replacing thymine),

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

and proteins as strings over an alphabet of 20, corresponding to the 20 amino acid residues.

The basic string algorithmic problems that develop in computational molecular biology are:

- Exact pattern matching: given a pattern $P$ and a text $T$ to locate the occurrences of $P$ into $T$
- Approximate pattern matching: given a pattern $P$, a text $T$, a similarity metric distance function $d()$, and a threshold parameter $k$ to locate all positions $i$ and $j$ such that $d(P, T_{i \cdots j}) \leq k$
- Sequence alignment: given two string sequences, $T_1$ and $T_2$, try to find the best alignment between the two sequences according to various criteria. The alignment can be either local or global. A special case of this problem, which has great biological significance, is the longest common subsequence problem in which we try to locate the longest subsequence that is common to both sequences
- Multiple approximate and exact pattern matching in which more than two strings are involved into the computation
- String clustering: given a set of stings, cluster them into a set of clusters according to the distance between the involved strings; this problem has great biological significance because DNA sequence clustering and assembling overlapping DNA sequences are critical operations when extracting useful biological knowledge
- Efficient implementation of indexing techniques for storing and retrieving information from biological databases

Besides these classical string algorithmic problems, there are also applications that demand the processing of strings whose form deviates from the classical definition. The most known category of such variations are the *weighted* strings that are used to model molecular weighted sequences [54]. A molecular weighted sequence is a molecular sequence (a sequence of either nucleotides or amino acids) in which in every position can be stored a set of characters each having a certain weight assigned. This weight can model either the probability of appearance or the stability of the character's contribution to the molecular complex. These sequences appear in applications concerning the DNA assembly process or in the modeling of the binding sites of regulatory proteins. In the first case, the DNA must be divided into many short strings that are sequenced separately and then are used to assemble the sequence of the full string; this reassembling introduces a degree of uncertainty that initially was expressed with the use of the "don't care" character denoted as "$*$," which has the property of matching against any symbol in the given alphabet. It is possible, though, that scientists are able to be more exact in their modeling and determine the probability of a certain character to appear at a position; j then a position that previously was characterized as a wild card is replaced by a probability of appearance for each of the characters of the alphabet and such a sequence is modeled as a *weighted sequence*. In the second case, when a molecular weighted sequence models the binding site of a regulatory protein, each base in a candidate motif instance makes some positive, negative, or neutral contribution to the binding stability of the DNA–protein complex [37, 65], and the weights assigned to each character can be thought of as modeling those effects. If the sum of the individual contributions is greater than a threshold, then the DNA–protein complex can be considered stable enough to be functional.

A related notion is the notion of an *indeterminate* or (equivalently in the scientific literature) a *degenerate* string. This specific term refers to strings in which each position contains a set of characters instead of a simple character; in these strings, the match operation is replaced by the subset operation. The need for processing efficiently such strings is driven by applications in computational biology cryptanalysis and musicology [76, 45]. In computational biology, DNA sequences still may be considered to match each other if letter A (respectively, C) is juxtaposed with letter T (respectively, G); moreover, indeterminate strings can model effectively polymorphism in protein coding regions. In cryptanalysis, undecoded symbols can be modeled as the set or characters that are candidates for the specific position, whereas in music, single notes may match chords or a set of notes.

Perhaps the most representative application of indeterminate strings is haplotype inference [42, 63]. A haplotype is a DNA sequence that has been inherited by one parent. A description of the data from a single copy is called a haplotype, whereas a description of the mixed data on the two copies is called a genotype. The underlying data that form a haplotype is either the full DNA sequence in the region or, more commonly, is the values of only DNA positions that are single nucleotide polymorphisms (SNPs). Given an input set $D$ of $n$ genotype vectors, a solution to the haplotype inference problem is a set of $n$ pairs of binary strings one pair for each genotype; for any genotype $g$, the associated binary vectors $v_1$, $v_2$ must be a "feasible resolution" of $g$ into two haplotypes that could explain how $g$ was created. Several algorithms have been proposed for the haplotype inference problem such as those based on the "pure parsimony criteria," greedy heuristics such as "Clarks rule," Expectation Maximization (EM)-based algorithms, and algorithms for inferring haplotypes from a set of Trios [42, 63]. Indexing all possible haplotypes that can be inferred from $D$ as well as gathering statistical information about them can be used to accelerate these haplotype inference algorithms. Moreover, as new biological data are being acquired at phenomenal rates, biological datasets have become too large to be readily accessible for homology searches, mining adequate modeling, and integrative understanding. Scalable and integrative tools that access and analyze these valuable data need to be developed. The new generation of databases have to (i) encompass terabytes of data, often local and proprietary; (ii) answer queries involving large and complex inputs such as a complete genome; and (iii) handle highly complex queries that access more than one dataset. These queries demand the efficient design of string indexing data structures in external memory; the most prominent of these structures are: the string B-tree of Ferragina and Grossi [30], the cache oblivious string dictionaries of Brodal and Fagerberg [15], the cache-oblivious string B-trees [14], and various heuristic techniques for externalizing the suffix tree [28].

In the sequel, (i) we will present the main string indexing data structures (suffix trees and suffix arrays ), (ii) we will present the main indexing structures for weighted and indeterminate strings, and (iii) we will present the main external memory string indexing structures.

## 1.2 MAIN STRING INDEXING DATA STRUCTURES

In this subsection, we will present the two main string indexing structures, suffix trees [92] and suffix arrays [67], and depict their special characteristics and capabilities.

### 1.2.1 Suffix Trees

The suffix tree is the most popular data structure for full string indexing, which was presented by Weiner in 1973 [92]. It is considered the oldest and most studied data structure in the area that, besides answering effectively to the pattern matching problem, can be used for the efficient handling of a plethora of string problems (see [41] for a set of applications in the area of computational molecular biology). More formally, the suffix tree $ST_T$ of a text $T$ is a compact digital search tree (trie) that contains all suffixes of $T$ as keys. It is assumed that before building the suffix tree, the text $T$ gets padded with an artificial character—the \$ character, which does not belong in the alphabet $\Sigma$ from which $T$ was formed. This assumption is used to guarantee that every suffix is stored to a distinct leaf of the tree (that is, no suffix is a prefix of another). The leaf of the tree that corresponds to the suffix $T_{i\cdots n}\$$ stores the integer $i$.

The suffix tree has the following structural properties that are induced by its definition:

- There are $n$ leaves—a leaf for each suffix of $T$. The concatenation of the substrings at the edges, which we traverse when moving from the root to the leaf that stores $i$, forms the suffix $T_{i\cdots n}$.
- Consider two different suffixes of $T$, $T_{i\cdots n} = xa$ and $T_{j\cdots n} = xb$, that share a common prefix $x$. In the suffix tree, the two leaves that correspond to the two suffixes have a common ancestor $u$ for whom the concatenation of the substrings at the edges that we traverse, moving from the root to the $u$, forms the common prefix $x$. This also can be phrased in a different way. For every internal node $u$ of a suffix tree, *all* suffixes that correspond to the leaves of its subtree share a common prefix $x$ that is represented from the edges of the path from the root to $u$. The substring that is created from the concatenation of the substrings of the edges traversed when moving from the root to $u$ is called the *path label* of the node $u$.

A lot of sequential algorithms have been proposed for building a suffix tree in linear time. The algorithms provided in [92, 70, 88] are based on the assumption that the strings have been formed from alphabets of constant size and basically are based on similar concepts. On the other hand, and for large alphabets (where we cannot ignore a time cost similar to its size), an elegant linear time algorithm has been proposed in [27]. Finally, in [10, 43, 81] parallel algorithms have been presented for the CRCW PRAM parallel model of computation.

Concerning implementation, a suffix tree $ST_T$ for a string $T$ of $n$ characters will have at most $2n - 1$ nodes and $2n - 2$ edges. The edges of a node can be stored either in a linear list (unsorted or sorted according to the first character of the edge label) or in an array of size $|\Sigma|$. In the first case (space-efficient implementation), a node can be traversed in $O(|\Sigma|)$ time, whereas in the second case, a node can be transversed in $O(1)$ time (though the space complexity for large alphabets is clearly worse). Between these two extreme choices of implementation, we can choose other alternatives as search trees or hash tables. The most efficient implementation, especially in the average case, is based in the use of a hash table. In [70], it the usage of the hashing scheme of Lampson [61], is proposed which belongs to the class of hash functions with chaining.

The suffix tree data structure can be extended to store the suffixes of more than one strings. In this case, we talk about the *generalized suffix tree*. More formally, a generalized suffix tree (GST) $(\{T_1, T_2, \cdots T_k\})$ of a set of strings $\{T_1, T_2, \cdots T_k\}$ is the compact trie that contains all suffixes of these strings as keys. For the construction of a generalized suffix tree, we can use the known algorithms for constructing suffix trees by superimposing the suffixes of different strings in the same structure; when having completed the insertion of the suffixes for a string, the procedure is continued for the next string by beginning from the root. A generalized suffix tree occupies $O(|T_1| + |T_2| + \cdots + |T_k|)$ space, and it can be built in $O(|T_1| + |T_2| + \cdots + |T_k|)$ time [41].

Concerning applications, let us consider the pattern matching problem and see how the suffix tree deals with the specific problem. Consider a string $T$ for which we have built the suffix tree $ST_T$ and suppose that we want to locate the positions within it where a pattern $P$ appears. By starting from the root of $ST_T$, we follow the path that is defined by $P$. After the $i$-th step of this procedure, if we are at an internal node and we have matched the $i$ leftmost characters of $P$, then we follow the outgoing edge that starts with the character $P_{i+1}$, whereas if we are at the interior of the edge, then we test whether the next character at the edge is equal to $P_{i+1}$. If this traversal from the root to the leaves finishes by matching successfully all $|P|$ characters of the pattern, then according to the aforementioned properties, the suffixes that correspond to the subtree below the point where the pattern matching procedure ended, share the pattern $P$ as a common prefix. Hence, the requested pattern appears at the positions that correspond to the leaves of that subtree. If the match procedure from the root to the leaf finishes before accessing all characters of the pattern, then no suffixes of $T$ can have the pattern $P$ as prefix; hence, the pattern does not appear anywhere inside the text. As a conclusion and with the assumption that in every internal node the edge that will be followed is being chosen in constant time, at most $|P|$ comparisons with the characters of the pattern are performed and the time complexity totals $(|P| + \alpha)$, where $\alpha$ is the size of the answer.

The suffix tree can answer to numerous other questions optimally besides performing pattern matching efficiently. The interested reader can consult [41] and the respective chapters in [72] and [3]. Some characteristic applications of the suffix tree are the *longest repeated substring* and the *longest common substring* (LCS) problems. In the longest repeated substring problem, we seek the longest substring of a text $T$ that appears in $T$ more than once. Initially, we built the suffix tree $ST_T$ in $O(|T|)$ time, and then by performing a traversal of the nodes of the suffix tree, we compute for every node the number of characters of the string from the root to the node. Then, we locate the internal node with the label of maximum length; the positions that are stored in the leaves of the subtrees below that node are the positions where the longest repeated substring appears. In the LCS problem, we search for the longest common substring of two strings $T_1$ and $T_2$. Initially and in time $O(|T_1| + |T_2|)$, we construct the generalized suffix tree $gST(\{T_1, T_2\})$ of the two sequences. In this generalized tree, some leaves will store suffixes of the one string, some of the other, and some will store suffixes of both strings. We traverse all nodes of the tree, and we compute for every node the number of the characters from the root to it; by a similar traversal, we mark the nodes in whose subtrees are stored leaves of both strings. Then to get our reply, we simply have to select the internal marked node with the path label of maximum length. Then the positions that correspond to the leaves of the corresponding subtrees are the positions where the longest common substring appears. With a similar linear time algorithmic procedure, we can locate the longest common substring between a set of more than two strings.

Concluding the suffix tree, is the main and better representative for data structures for full text indexing. The cost for this enhanced functionality is the extra space complexity. There are cases in which the required space can be 25 times more than the indexed data. This fact and the poor behavior when being transferred in secondary memory restricts the use of suffix trees in applications that are limited in the main memory of a computer system.

Optimizations of the suffix tree structure to face these disadvantages were undertaken by McCreight [70] and more recently by Kurtz [62]. Kurtz reduced the RAM required to around 20 bytes per input character indexed on the worst case and to 10,1 bytes per input character on average. Compact encodings of the suffix tree based on a binary representation of the text have been investigated by Munro and Clark [20] Munro *et al.* [73] and Anderson *et al.* [7]. There are also other works concerning efficient compression of the suffix tree; the interested reader should consult [32, 38, 39, 73, 80] for further details on this interesting algorithmic area.

### 1.2.2 Suffix Arrays

The suffix arrays have been introduced in [67] and constitute the main alternative full text indexing data structure as compared with the suffix tree. The main advantages of the suffix array are its small memory requirements, its implementation simplicity, and the fact that the time complexities for constructing and query answering are independent from the size of the alphabet. Its main disadvantages are that the query time is usually larger than the respective query time of the suffix tree and that the range of applications where it can be used is smaller than the range of applications of the suffix tree.

More formally, a suffix array $SA_T$ for a string $T$ of $n$ characters is an array that stores the suffixes of $T$ in lexicographic order. That is, in every position $i$ of the suffix array, the starting position $j$ of the suffix $T_{j\dots n}$ $(SA_T[i] = j)$ is stored in such a way that the suffixes that are lexicographically smaller than $T_{j\cdots n}$ are located in positions smaller than $i$, whereas the suffixes that are lexicographically larger than $T_{j\cdots n}$ are located in positions larger than $i$. Hence, we get $T_{SA_T[1]\cdots n} <_L T_{SA_T[2]\cdots n} <_L \cdots <_L T_{SA_T[n]\cdots n}$ where $<_L$ designates the lexicographic order. Because suffix arrays store the suffixes of $T$ lexicographically ordered they have the following property: suppose that the suffixes, located at positions $i$, $j$, with $i < j$ have a common prefix $x$, that is LCP $({}_{SA_T[i]\cdots n}, {}_{SA_T[j]\cdots n}) = x$. Then all suffixes $T_{SA_T[w]\cdots n}$ that are located in positions $i \leq w \leq j$ have $x$ as a prefix.

Because the suffix array is basically an array of $n$ elements without the need for extra pointers, its space requirements are significantly smaller (in terms of constant factors) from the respective space requirements that characterize the suffix trees. However, the use of suffix array without extra information does not permit efficient searching. To understand this concept, let us explain how the suffix tree can solve the problem of exact pattern matching of a pattern $P$ into a text $T$. To accomplish the search, we need to locate two positions $i$, $j$ with $i \leq j$ for which the following holds: the first $|P|$ characters of the suffixes at position $j$ are lexicographically smaller or equal from the pattern (that is $T_{SA_T[j]\cdots SA_T[j]+|P|} \leq_L P$), and $j$ is the maximum position with this property, whereas the first $|P|$ characters of the suffix at position $i$ are lexicographically larger or are more equal than the pattern (that is $P \leq_L T_{SA_T[i]\cdots SA_T[i]+|P|}$) and $i$ is smaller with that property. According to that, the suffixes that correspond to positions $i$, $j$, and all intermediate positions have $P$ as a prefix. Consequently, the places where $P$ appears in $T$ can be located by finding the two extreme positions $i$, $j$ in the suffix array and then scanning the intermediate positions. To locate the extreme positions, a binary search needs to be executed on the suffix array in which at each step of the search procedure, $|P|$ comparisons are needed and then the procedure moves right or left. Hence, the problem of pattern matching by using the suffix arrays is solved in $(|P| \log n + \alpha)$ time, where $\alpha$ is the size of the answer. This time complexity can be reduced significantly to $(|P| + \log n + \alpha)$ if we use two more arrays of $n - 2$ elements containing precomputed information; with the help of these elements, it is possible in every repetition of the binary search procedure, not to execute all $|P|$ comparisons that correspond to the middle of the active segment. In particular, suppose that the binary search procedure is in an interval $[L, R]$ of the suffix array, and we seek to compute the value $m = \text{LCP}(P, T_{SA_T[M]\cdots n})$ for the middle of the search interval. We suppose that the values $l = \text{LCP}(P, T_{SA_T[L]\cdots n})$ and $r = \text{LCP}(P, T_{SA_T[R]\cdots n})$ have been computed in a previous repetition of the binary search. The first remark that can be made is that we do not have to perform all $|P|$ comparisons from the beginning because of the basic property of the suffix array $m \geq \min\{l, r\}$; hence, the comparisons can continue from position $m + 1$, and hence, it is possible to save $\min\{l, r\}$ comparisons. However, despite this improvement, there are scenarios in which the order of the total complexity does not change. Suppose now that we have as extra information the longest common prefix of the suffixes of the left edge $L$ of the search interval, with the suffix at the middle and the longest common prefix of the right edge $R$ of the search interval with the suffix of the middle. Let us symbolize them as $x = \text{LCP}(T_{SA_T[L]\cdots n}, T_{SA_T[M]\cdots n})$ and $x' = \text{LCP}(T_{SA_T[M]\cdots n}, {}_{SA_T[R]\cdots n})$, respectively. Let us suppose that $l \geq r$, and hence, we have the following, depending on whether $x$ is the longest common prefix of the left edge with the medium, or is largest, smaller, or equal to $l$:

- If $x > l$, then because $L$ has the first $l$ characters equal to $P$ and the first $x$ equal to the suffix at position $M$ and it holds $x > l$, the $l + 1$-st character of the suffix at position $M$ does not match with the $l + 1$-st character of $P$. Hence, according to the basic property of the suffix array, no common prefix exists with $P$ to the left side of $M$. Hence, we choose $[M, R]$ as the new interval.
- If $x < l$, then because $P$ matches with the $l$ characters of $L$ and with the $x$ characters of the middle suffix, we will have a nonmatching of the middle suffix at position $x + 1$. Hence, a larger prefix of $P$ must exist in the left interval, and hence, we will choose $[L, M]$ as the new search interval.
- If $x$ equals $l$, then we cannot deduce that the longest common prefix with $P$ is in the left or the right interval. Hence, by a character to character comparison, we extend the common prefix (if it can be extended) beyond the position $l$. If we perform $\Delta h$ successful comparisons, then the common prefix of the suffix of $M$ with $P$ will have length $l + \Delta h$. The failure in matching at position $l + \Delta h + 1$ guides us left or right depending on whether the character of the corresponding position at $P$ is lexicographically smaller or larger than the respective position at the middle suffix.

Hence, every one of the $O(\log n)$ steps of the binary search either performs a constant number of operations (cases $x > l$ or $x < l$) or performs $\Delta h$ comparisons (case $x = l$). The sum of comparisons in the last case does not exceed $|P|$ because the middle chosen element will be one of the extreme elements in the next repetition (its value is continuously increasing). Hence, the problem of pattern matching is being solved in $(|P| + \log n)$ time.

Concerning the needed space consumption, the improved time complexity is achieved by using the $\text{LCP}(T_{SA_T[L]\cdots n}, T_{SA_T[M]\cdots n})$ and $\text{LCP}(T_{SA_T[M]\cdots n}, T_{SA_T[R]\cdots n})$ values as precomputed information for every possible interval that can exist during binary searching. The number of different intervals is $n - 2$ because the role of middle elements can be played by all elements, except the first and the last. Hence, one array stores the values of the left extreme for every possible middle element, whereas the other array stores the values of the right extreme. The existing suffix array algorithms precompute in the arrays $\text{LCP}[i] = \text{LCP}(T_{SA_T[i]\cdots n}, T_{SA_T[i+1]\cdots n})$ for $i = 1 \cdots n$ in linear time. By using the relationship $\text{LCP}(L, R) = \min\{\text{LCP}(L, M), \text{LCP}(M, R)\}$ from this array, we can create the LCP values for all possible intervals of binary searching. Concluding the enhanced suffix array construction occupies approximately $5n$ space (see also chapter 29 of [3]), which is less than the space complexity of the suffix tree. The time to solve a pattern matching problem is $O(|P| + \log n)$, which can be compared with the time required for the query replying of a suffix tree $O(|\Sigma|P)$ (in the implementation that is space effective) or $O(|P|)$ (in the implementation that is time effective). The construction algorithm that has been presented initially in [67] was not linear but needed $O(n \log n)$ time. A linear time procedure can be envisaged by simply constructing (in linear time) a suffix tree and then transforming it to the respective suffix array. This transformation is possible by traversing lexicographically the suffix tree in linear time and then computing the arrays LCP in linear time by nearest common ancestor queries. This procedure takes linear time but cancels the main advantage of the suffix tree, which is the small space consumption. In 2003, three papers were published [58, 75, 26] that describe a worst-case linear time construction of a suffix array without the need of an initial construction of the respective suffix tree. Other algorithms for constructing suffix arrays can be found in [17, 36, 47, 55, 68]. Moreover, a recent line of research concerns compressed suffix arrays [46, 38, 39, 35].

Concerning applications, the main weakness of the suffix array in comparison with the suffix tree data structure is that the range of application in which it can be used is limited. To resolve this handicap, in [59], a method was presented that combined the information of a suffix array with the LCP information, which simulates the postorder traversal in the equivalent suffix tree of the string, thus providing the so-called *virtual suffix tree*. This simulation (which was extended in [35] with a space efficient variant) gives the ability for some of the suffix tree applications whose algorithmic procedure is based in the bottom-up traversal of the suffix tree to be executed with some extra changes in the suffix array.

The suffix array table is being traversed from left to right, and an auxiliary stack is being used. Initially, the stack contains the root and $\mathrm{LCP}[1] = \mathrm{LCP}(T_{SA_T[1]\cdots n}, T_{SA_T[2]\cdots n}) \geq 0$. If this value is equal to zero, then the two leftmost leaf-suffixes have a minimum common ancestor in the root, and hence, during the implicit postorder traversal, we process the first and then the second element. If the value is greater than zero, then an internal node exists that is being inserted in the stack. More generally, during step $i$, if $\mathrm{LCP}[i] = (T_{SA_T[i]\cdots n}, T_{SA_T[i+1]\cdots n})$ is larger than the depth of the node $u$ at the top of the stack (that is, the length of the path label $L(u)$), then between the $i$-th leaf/suffix and the next, a deeper node exists that will be inserted in the stack; otherwise, the value $\mathrm{LCP}[i]$ is smaller than the depth of node $u$, and the minimum common ancestor is higher in the path from $u$ to the root. In this case, the stack is emptied until a node is located with smaller depth, and the first case is applied. Based on this described procedure, a node is inserted in the stack when it is seen during the top-down traversal, whereas it is removed from the stack when it is faced moving bottom up for the last time. Because in every step of the method, we either add a node in the stack or we have several deletions from the stack, every node is inserted and deleted from the stack once, and the whole procedure needs $O(n)$ time. In [1], other combinations of the suffix array with additional information were provided (the so-called *enhanced suffix array*), and additional applications were described.

## 1.3 INDEX STRUCTURES FOR WEIGHTED STRINGS

The notion of the weighted sequence extends the notion of a regular string by permitting in each position the appearance of more than one character, each with a certain probability. In the biological scientific literature weighted sequences also are called position weight matrices (PWM). More formally, a weighted word $w = w_1 w_2 \cdots w_n$ is a sequence of positions with each position $w_i$ consisting of a set of couples, of the form $(s, \pi_i(s))$, with $\pi_i(s)$ being the probability of having the character $s$ at position $i$. For every position $w_i$, $1 \leq i \leq n$, $\sum \pi_i(s) = 1$.

For example, if we consider the DNA alphabet $\Sigma = \{A, C, G, T\}$, then the word $w =$ [(A,0.25), (C,0.5), (G,0.25), (T,0)][(A,1), (C,0), (G,0), (T,0)] [(A,1), (C,0), (G,0), (T,0)] represents a word having three letters; the first one is either A,C,G with probabilities of appearance of 0.25, 0.5, and 0.25, respectively; the second one is always A, whereas the third letter is necessarily an A because its probability of presence is equal to 1. The probability of presence of a subword either can be defined to be the cumulative probability, which is calculated by multiplying the relative probabilities of appearance of each character in every position, or it can be defined to be the average probability.

There have been published works in the scientific literature [19, 5, 6, 54] concerning the processing of string sequences; we will refer to these works giving more emphasis to the structure presented in [54]. In [19], a set of efficient algorithms were presented for string problems developing in the computational biology area. In particular, assume that we deal with a weighted sequence $X$ of length $n$ and with a pattern $p$ of length $m$, then (i) the occurrences of $p$ in $X$ can be located in $O((n + m) \log m)$ time and linear space; the solution works for both the multiplicative and the average model of probability estimation, although it can be extended also to handle the appearance of gaps; (ii) the set of repetitions and the set of covers (of length $m$) in the weighted sequence can be computed in $O(n \log m)$ time. In [6] and for the multiplicative model of probability estimation the problem of approximately matching a pattern in a weighted sequence was addressed. In particular, two alternative definitions were given for the *Hamming distance* and two alternative definitions for the *edit distance* in weighted sequences with the aim of capturing the aspects of various applications. The authors presented algorithms that compute the two versions of the Hamming distance in time $O(n\sqrt{m \log m})$, where the length of the weighted text is $n$, and $m$ is the pattern length; the algorithms are based in the application of nontrivial bounded divide-and-conquer algorithms coupled with some insights on weighted sequences. The two versions of the edit distance problem were solved by applying dynamic programming algorithm with the first version being solved in $O(nm)$ time and the other version in $O(nm^2)$ time. Finally, the authors extended the notion of weighted matching in infinite alphabets and showed that exact weighted matching can be computed in $O(s \log^2 s)$

time, where $s$ is the number of text symbols with nonzero probability, and they also proved that the weighted Hamming distance over infinite alphabets can be computed in $\min(O(kn\sqrt{s} + s^{3/2}\log^2 s), O(s^{4/3}m^{1/3}\log s))$, where $m$ is the length of the pattern. In [5], a different approach was followed, and a transformation was proved between weighted matching and *property matching* in strings; the pattern matching with properties (property matching for short) was introduced in the specific paper in which pattern matching with properties involves a string matching between the pattern and the text and the requirement that the text part satisfies some property. The aforementioned reduction allows off-the-self solutions to numerous weighted matching problems (some were not handled in the previously published literature) such as scaled matching, swapped matching, pattern matching with swaps, parameterized matching, dictionary matching, and the indexing problem. All presented results are enabled by a reduction of weighted matching to property matching that creates an ordinary text of length $O(n(\frac{1}{\epsilon})^2 \log \frac{1}{\epsilon})$ for the weighted matching problem of length $n$ and the desired probability of appearance $\epsilon$. Based on this reduction, all pattern matching problems that can be solved in ordinary text can have their weighted versions solved with the time degradation of the reduction.

Finally in [54], a data structure was presented for storing weighted sequences that can be considered the appropriate generalization of the suffix tree structure to handle weighted sequences. A resemblance exists between this structure and the work related to *regulatory motifs* [71, 66, 83, 60] and *probabilistic suffix trees* [78, 82, 69]. Regulatory motifs characterize short sequences of DNA and determine the timing location and level of gene expression, and the approaches extracting regulatory motifs can be divided into two categories: those that exploit word-counting heuristics [57, 69] and those based on the use of probabilistic models [40, 48, 64, 79, 85, 87]; in the second category of approaches, the motifs are represented by position probabilistic matrices, whereas the remainder of the sequences are represented by background models. The probabilistic or *prediction* suffix tree is basically a stochastic model that employs a suffix tree as its index structure to represent compactly the conditional probabilities distribution for a cluster of sequences. Each node of a probabilistic suffix tree is associated with a probability vector that stores the probability distribution for the next symbol given the label of the node as the preceding segment, and algorithms that use probabilistic suffix trees to process regulatory motifs can be found in [82, 69]. However, the probabilistic suffix tree is inefficient for efficiently handling weighted sequences, which is why the weighted suffix tree was introduced; however, it could be possible for a suitable combination of the two structures to be effective to handle both problem categories.

The main idea behind the weighted suffix tree data structure is to construct the suffix tree for the sequence incorporating the notion of probability of appearance for each suffix; that is, for every suffix $x[i \cdots n]$, we store in a set of leaves labeled $S_i$ the first $l$ characters so that $\pi(x_i \cdots x_{i+l-1}) \geq 1/k$. In more detail, for every suffix starting at position $i$, we define a list of possible weighted factors (not suffixes because we may not eventually store the entire suffix) so that the probability of appearance for each one of them is greater than $1/k$; here, $k$ is a user-defined parameter that is used to denote substrings that are considered valid.

The formal definition as provided in [54] is that the weighted suffix tree (WST) of a weighted sequence $S$ (denoted as WST($S$)) is the compacted trie of all weighted factors starting within each suffix $S_i$ of $S$ having a probability of appearance greater than 1/k. The leaf $u$ of WST($S$) is labeled with index $i$ if $L(u) = S_{i,j}[i \cdots n]$ and $\pi(S_{i,j}[i \cdots n]) \geq 1/k$, where $j > 0$ denotes the $j$-th weighted factor starting at position $i$. $L(u)$ denotes the path label of node $u$ in WST($S$) and results by concatenating the edge labels along the path from the root to $u$. $D(u) = |L(u)|$ is the string-depth of $u$, whereas LL($u$) is defined as the leaf list of the subtree below u. LL($u$) $= \emptyset$ if $u$ is a leaf.

It can be proven that the time and space complexity of constructing a WST is linear to the length of the weighted sequence.

The WST is endowed with most of the sequence manipulation capabilities of the generalized suffix tree, that is:

- Exact pattern matching: Let $P$ and $T$ be the pattern and the weighted sequence, respectively. Initially, the weighted suffix tree is built for $T$, and if the pattern $P$ is weighted, too, then it is broken into solid subwords; for each of these subwords, the respective path is spelled by moving from the root of the tree until an internal node is reached then all leaves descending from this node are reported. The time complexity of the procedure is $O(m + n + a)$, where $m$ and $n$ are the sizes of the pattern and the text, respectively, and $a$ is the answer size.
- Finding repetitions in weighted sequences: It is possible to compute all repetitions in a given weighted sequence, with each repetition having a probability of appearance greater than $1/k$; initially, the respective weighted suffix tree is constructed, and then the weighed suffix tree is traversed with a depth-first traversal, during which a leaf list is kept for each internal node. The elements of a leaf list are reported if the size of the list exceeds two; in total, the problem is solved in $O(n + a)$ time, where $n$ is the sequence length and $a$ is the answer size.
- Longest common substring of weighted sequences: The generalized weighted suffix tree is built for two weighted sequences, $w_1$ and $w_2$, and then the internal node with the greatest depth is located; the path label of this node corresponds to the longest weighted subsequence of the two weighted strings. The time complexity of the procedure is equal to $O(n_1 + n_2)$, with $n_1$ and $n_2$ being the sizes of $w_1$ and $w_2$, respectively.

## 1.4 INDEX STRUCTURES FOR INDETERMINATE STRINGS

*Indeterminate* or (equivalently in the scientific literature) *degenerate* strings are strings that in each position contain a set of characters instead of a simple character. The simplest form of indeterminate string is one in which indeterminate positions can contain only a do-not-care letter that is a letter “$*$,” which matches any letter in the alphabet on which $x$ is defined. In 1974, an algorithm was described [33] for computing all occurrences of a pattern in a text where both the pattern and the text can contain do-not-care characters, but although efficient in theory, the algorithm was not useful in practice; the same remark also holds for the algorithms presented in [2]. In [11, 93], the shift-or technique (a bit-mapping technique for pattern matching) was applied to find matches in indeterminate strings. In [51], an easily implemented average case $O(n)$ time algorithm was proposed for computing all periods of every prefix of a string with do-not-care characters. In [44], this work was extended by distinguishing two forms of indeterminate match ("quantum" and "deterministic") and by refining the definition of indeterminate letters so that they can be restricted to matching only with specified subsets of $\Sigma$ rather than with every letter of $\Sigma$. More formally, a "quantum" match allows an indeterminate letter to match with two or more distinct letters during a single matching process, whereas a "deterministic" match restricts each indeterminate letter to a single match.

These works were continued by researchers in [9, 8, 50, 52, 53, 76, 45], in which a set of algorithms were presented for computing repetitive structures, computing covers, computing longest common subsequences, and performing approximating and exact pattern matching; some of them improved the aforementioned previous constructions. From these structures, special emphasis should be given to the works in [76] and [45] because they fell in the focus of interest of this chapter. In particular, in [45], efficient practical algorithms were provided for pattern matching on indeterminate strings where indeterminacy may be determined either as "quantum" or "deterministic"; the algorithms are based on the Sunday variant of the Boyer–Moore pattern matching algorithm and are applied more generally to all variants of Boyer–Moore (such as Horspool's) that depend only on the calculation of the "rightmost shift" array. It is assumed that $\Sigma$ is indexed being essentially an integer alphabet. Moreover, three pattern-matching models are considered in increasing order of sophistication: (i) the only indeterminate letter permitted is the do-not-care character, whose occurrences may be either in the pattern or in the text, (ii) arbitrary indeterminate letters can occur but only in the pattern, (iii) indeterminate letters can occur in both the pattern and the text. In [76], and asymptotically faster algorithms were presented for finding patterns in which either the pattern or the text can be *degenerate* but not both. The algorithms for DNA and RNA sequences work in $O(n \log m)$ time, where $n$ and $m$ are the lengths of the text and the pattern, respectively. Efficient implementations also are provided that work in $O(n + m + n \lceil m/w \rceil + \lceil n/w \rceil)$ time, where $w$ is the word size; as can be seen, for small sizes of the text and the pattern, the algorithms work in linear time. Finally it also is shown how their approach can be used to solve the distributed pattern matching problem.

Concerning indexing structures, there are some results that can be divided into two categories, one based on the use of compressed tries and the other based on the used of finite automata.

Concerning results in the first category in [63], the dictionary matching problem was considered in which the dictionary $D$ consists of $n$ indeterminate strings and the query $p$ is a string over the given alphabet $\Sigma$. A string $p$ matches a stored indeterminate string $s_i$ if $|p| = |s_i|$ and $p[j] \in s_i[j]$ for every $1 \leq j \leq |p|$. The goal is to preprocess $D$ for queries that search for the occurrence of pattern $p$ in $D$ and count the number of appearances of $p$ in $D$.

Let $m$ denote the length of the longest string in $D$ and let $D'$ be the set of all patterns that appear in $D$. For example, if $D$ contains a single indeterminate string $cd\{a, b\}c\{a, b\}$, then $D' = \{cdaca, cdacb, cdbca, cdbcb\}$. The data structure is basically a compressed trie of $D'$ that can be constructed naively in $O(|\Sigma|^k nm)$ time and $O(|\Sigma||D'|)$ space, assuming every $s \in D$ has at most $k$ locations in which $|s[i]| > 1$. With this structure, a query time of $O(|p|)$ can be supported for a pattern $p$ plus a time complexity equal to the size of the output. Using techniques presented in [22], the structure can be modified to solve the problem in $O(nm \log(nm) + n(c_1 \log n)^{k+1}/k!)$ preprocessing time, and $O(m + (c_2 \log n)^k \log\log n)$ query time ($c_1$ and $c_2$ are constants); this approach is worse than the trie approach for small values of $\Sigma$.

In [63], two faster constructions of the trie have been presented. The first construction is based on the divide-and-conquer paradigm and requires $O(nm + |\Sigma|^k n \log n)$ preprocessing time, whereas the second construction uses ideas introduced in [4] for text fingerprinting and requires $O(nm + |\Sigma|^k n \log m)$ preprocessing time. The space complexity is $O(|\Sigma||D'|)$, and it can be reduced to $O(|D'|)$ by using the suffix tray [23] ideas. The query time becomes $O(|p| + \log\log|\Sigma|)$, and it is also possible by cutting the dictionary strings and constructing two tries to obtain $O(nm + |\Sigma|^k n + |\Sigma|^{k/2} n \log(\min\{n, m\}))$ preprocessing time at the cost of $O(|p| \log\log|\Sigma| + \min\{|p|, \log|D'|\} \log\log|D'|) = O(|p| \log\log|\Sigma| + \min\{|p|, \log(|\Sigma|^k n)\} \log\log(|\Sigma|^k n))$ query time. The first two constructions can calculate the number of appearances in $D$ of each pattern in $D'$, a knowledge that can be useful in a possible application of the structures to the Haplotype inference problem [63].

On the other hand, there are works based in the use of finite automata, which are based in indexing small factors (that is, substrings of small size). Indexing of short factors is a widely used and useful technique in stringology and bioinformatics, which has been used in the past to solve diverse text algorithmic problems. More analytically, in [90], the *generalized factor automaton* (GFA) was presented, which has the disadvantage that it cannot be used to index large texts because, experimentally, it tends to grow super-quadratically with respect to the length of the string. Later in [91], the truncated generalized factor automaton (TGFA) was presented that is basically a modification of GFA that indexes only factors with length not exceeding a given constant $k$ having at most a linear number of states. The problem with the specific algorithm is that it is based on the subset construction technique and inherits its space and time complexity that is a bottleneck of the algorithm when indexing very long text because the corresponding large Nondeterministic Finite Automaton needs to be determinized. Finally, in [34], an efficient on-line algorithm for the construction of the TGSA was presented, which enables the construction of TGSAs for degenerate strings of large sizes (order of Megabytes (MBs)). The proposed construction works in $O(n^2)$ time, where $n$ is the length of the input sequence. TGSA has, at most, a linear number of states with respect to the length of the text and enables the location of the list $occ(u)$ of all occurrences of the given pattern $u$ in the degenerate text in time $|u| + |\text{occ}(u)|$.

## 1.5 STRING DATA STRUCTURES IN MEMORY HIERARCHIES

We consider string data structures in external memory [89]. The design of such data structures is a necessity in the computational molecular biology area, as the datasets in various biological applications and the accumulated data in DNA sequence databases are grown exponentially, and it is not possible to have them in main memory; a characteristic number is provided in [74] where it is mentioned that in the GenBank, the DNA sequence database has crossed the 100 Gbp (bp stands for base pairs), with sequences from more than 165,000 organisms. The basic external memory model used to analyze the performance of the designed algorithms is the two-level memory model; in this model, the system memory is partitioned into a small but fast in access partition (the main memory with size M) and into a (theoretically unbounded) secondary part (the disk). Computations are performed by central processing unit (CPU) on data that reside in main memory while data transfer between memory and disk take place in contiguous pieces of size $B$ (the block size).

In string algorithmics, there are two lines of related research, one that focuses on transferring the main-memory-tailored design of the suffix tree and/or suffix array data structures to secondary memory and another that tries to envisage novel, external-memory-tailored data structures with the same functionality as the suffix tree.

In the first line of work, a plethora of published material exists dealing with the externalization of the suffix tree: [49, 74, 86, 16, 49, 12, 13, 18, 21, 56, 84, 86] and the suffix array [24, 25]. Most of these works suffer from various problems such as nonscalability, nonavailability of suffix links (that are necessary for the implementation of various operations) and nontolerance to data skew, and a few are the works that manage to face effectively these problems; from these works, we will present briefly the approach in [74]. More specifically, the authors in [74] present TRELLIS, an algorithm for constructing genome-scale suffix trees on disk with the following characteristics: (i) it is an $O(n^2)$ time and $O(n)$ space algorithm that consists of four main phases—the prefix creation phase, the partitioning phase, the merging phase, and the suffix link recovery phase; the novel idea of the algorithm lies in the use of variable length prefixes for effective disk partitioning and in a fast postconstruction phase for recovering the suffix links; (ii) it can scale effectively for very large DNA sequences with suffix links; (iii) it is shown experimentally that it outperforms most other constructions because it is depicted as faster than the other algorithms that construct the human genome suffix tree by a factor of 2–4 times; moreover, its query performance is between 2–15 times faster than existing methods with each query taken on the average between 0.01–0.06 seconds.

In the second line of research, string B-trees [30], cache oblivious string dictionaries [14], and the cache oblivious string B-tree [15] come into play.

The string B-tree [30] is an extension of the B-tree suitable for indexing strings with a functionality equivalent to the functionality of the suffix tree. More analytically, assume that we have to process a set $S$ of $n$ strings with a total number of $N$ characters and suppose that each of the strings is stored in a contiguous sequence of disk pages and is represented by its logical pointer to the external memory address of its first character. In the leaves of the string B-tree, we store the logical pointers of the strings lexicographically ordered, and the leaves also are linked together to form a bidirectional list. In every internal node $v$, we store as search keys the leftmost and the rightmost string stored in each of the node's children. Hence, if $v$ has $k$ children $c_i$, $(1 \leq i \leq k)$ then the keys stored in $v$ will be $K_v = \{L(c_1), R(c_1), L(c_2), R(c_2), \cdots, L(c_k), R(c_k)\}$, where $L(c_i)$ and $R(c_i)$ are the leftmost and the rightmost keys stored in the child $c_i$. The string B-tree in this form can answer prefix queries for a query string $P$. The total number of disk accesses will be $O(\frac{|P|}{B} \log_B |S| \log_2 B)$ I/Os because (i) $O(\log_2 B)$ accesses are needed in every internal node for locating the proper subtree via binary search, (ii) in every binary search step all characters of $P$ will need to be loaded from the disk, and thus, a total of $O(\frac{|P|}{B})$ disk I/O accesses are needed. The whole procedure is executed in every internal node moving from the root to the leaf, and hence, it is repeated $O(\log_B |S|)$ times.

The time complexity of the aforementioned procedure can be reduced by organizing the elements stored in each node of the string B-tree as a Patricia trie. A Patricia trie is a compact digital search tree (trie) that can store a set of $k$ strings in $O(k)$ space as follows: (i) a compacted trie of $O(k)$ nodes and edges is built on the $k$ strings; (ii) in each compacted trie node we store the length of the substring into it, and the substring that normally would label each edge is replaced by its first character. This construction gives the possibility to fit $O(B)$ strings into one node independently of the length of the strings and allows lexicographic searches by branching out from a node without further disk accesses.

By using Patricia tries for storing the strings in internal nodes, we see that we do not need binary search in each node, but it is possible to select the proper subtree in $O(\frac{|P|}{B})$ I/Os, and hence, the total time complexity of disk accesses when moving from the root to the leaf becomes $O(\frac{|P|}{B} \log_B |S|)$ I/Os. The query time complexity can be reduced further by a more careful search procedure that will take into account the observation that the longest common prefix that a query can have with the keys of a node is at least equal to the longest common prefix between the query and the keys stored in the parent of the node; in this case, the query time becomes $O(\frac{|P|}{B} + \log_B |S|)$ I/Os for completing the traversal from the root to the leaves. Concerning dynamic operations to insert/delete a string $T'$ a query initially is executed for locating the appropriate leaf position among the leaves of the string B-tree. If space exists for inserting the appropriate leaf, it is inserted; otherwise, the leaf gets split, and the father node is updated with appropriate pointers. The rest of the insertion and deletion procedure is similar to the balancing operations performed in the traditional B-tree with the difference that in the worst case the balancing operations can be propagated until the root of the tree, and hence, the total number of disk accesses will be bounded from above by $O(\frac{|T'|}{B} + \log_B |S|)$ I/Os.

The above lead to the following theorems:

**Theorem 1.1** *The string B-tree can store a set $S$ of $n$ strings with a total number of $N$ characters in $O(n/B)$ space (the index) plus $O(N/B)$ space (the characters* *of the string) so that the strings in S that have a query prefix P can be computed in* $O(\frac{|P|}{B} + \log_B n + \frac{a}{B})$ *I/Os, where a is the size of the answer. To insert or delete a string* $T'$ *in S,* $O(\frac{|T'|}{B} + \log_B n)$ *I/Os are needed.*

To use the string B-tree for efficient pattern matching, we should insert all suffixes of the involved strings, because in this case, prefix matching with a given pattern becomes equivalent to pattern matching with the given pattern. It can be proved that the produced structure will have the following properties:

**Theorem 1.2** *The string B-tree can store a set S of n strings with a total number of N characters in* $O(N/B)$ *space (the index) plus* $O(N/B)$ *space (the characters of the string) so that the strings in S that contain a given query pattern P can be computed in* $O(\frac{|P|+a}{B} + \log_B n)$ *I/Os, where a designates the size of the answer. To insert or delete a string of length m in S* $O(m \log_B(N + m))$ *I/Os are needed.*

The specific structure has been improved with two other structures [14, 15] that are valid for the cache oblivious model of computation. The cache-oblivious model is a generalization of the two-level I/O model to a multilevel memory model, by employing a simple trick: the algorithm is not allowed to know the value of $B$ and $M$, and thus, its functionality and working remains valid for any value of $B$ and $M$. In particular, in [15], a cache-oblivious string dictionary structure was presented supporting string prefix queries in $O(\log_B n + |P|/B)$ I/Os, where $P$ is the query string and $n$ is the number of stored strings. The dictionary can be constructed in $O(\text{Sort}(N))$ time where $N$ is the total number of characters in the input, and $\text{Sort}(N)$ is the number of I/Os needed for comparison-based sorting. The input as in the string B-tree can be either a set of strings to store or a single string for which all suffixes are to be stored; moreover, if it is given as a list of edges of the appropriate tree, then it also can accept a trie, a compressed trie, or a suffix tree. It is assumed that $M \geq B^{2+\delta}$. The aforementioned structure has the following two novel characteristics: (i) it uses the notion of the *giraffe tree* that provides an elegant linear space solution to the path traversal problem for trees in external memory; the giraffe trees permit the exploitation of redundancy because parts the path in the trie may be stored multiple times but with only a constant factor blowup in total space as the trie gets covered by them; (ii) it exploits a novel way for decomposing a trie into components and subcomponents based on judiciously balancing the progress in scanning the query pattern with the progress in reducing the number of strings left as matching candidates.

The aforementioned contribution was improved in [14] where a cache-oblivious string B-tree (COSB-tree) was presented that can search asymptotically optimal and insert/delete nearly optimal and can perform range queries with no extra disk seeks. An interesting characteristic of the structure is that it employs front compression to reduce the size of the stored set. In particular for a set $D$, assume that we denote by $||D||$ the sum of key lengths in $D$ and by $\text{front}(D)$ the size of the front-compressed $D$. The proposed structure has space complexity $O(\text{front}(D))$ and possesses the following performance characteristics: (i) insertion of a key $k$

requires $O(1 + ||k||(\log^2 \text{front}(D))/B + \log_B N)$ memory transfers with high probability (w.h.p.), (ii) searches and successor/predecessor queries for a key $k'$ require an optimal $O(1 + ||k'||/B + \log_B N)$ block transfers w.h.p. The result set $Q$ is returned in compressed representation and can be decompressed in additional $O(||Q||/B)$ memory transfers, which is optimal for front compression. Because COSB-trees, store all keys in order on disk range, queries involve no extra disk seeks.

An important component of the COSB-tree of independent interest is the front-compressed packed memory array (FC-PMA) data structure. The FC-PMA maintains a collection of strings $D$ stored in order with a modified front compression. As is shown in [14], the FC-PMA has the following properties: (i) for any $\epsilon$, the space usage of the FC-PMA can be set to $(1 + \epsilon)$ front$(D)$ while enabling a string $k$ to be reconstructed with $O(1 + ||k||/(\epsilon B))$ memory transfers, (ii) inserting and deleting a string $k$ into a FCPMA requires $O(||k||(\log^2 \text{front}(B))/(\epsilon B))$.

The interested reader can find a nice exposition of some of these plus other structures in [77, 28].

## 1.6 CONCLUSIONS

String indexing algorithms and data structures play a crucial role in the field of computational molecular biology, as most information is stored by means of symbol sequences. Storing, retrieving, and searching in this vast volume of information is a major task to have several specific queries and problems being solved efficiently. In this chapter, we have presented the main indexing structures in the area.

We conclude by noting that despite the great progress in the string indexing research field in the last decade, the frontiers need to move a little bit further by means of: (i) minimizing the volume of data with compression and searching in compressed files, (ii) minimizing the extent of the indexing structures by compressing them, too [29], (iii) building and placing the indexing structures cache obliviously to minimize the cache misses [31], and (iv) building the indexing structures efficiently in parallel, using the model multiprocessor machines and operating systems.

## REFERENCES

1. M. Abouelhoda, S. Kurtz, and E. Ohlebusch. Enhanced suffix arrays and applications. In S. Aluro, editor, *Handbook of Computational Molecular Biology*. Chapman and Hall, New York, 2006.
2. K. Abrahamson. Generalized string matching. *SIAM J Comput*, 16(6):1039–1051, 1987.
3. S. Aluru. *Handbook of Computational Molecular Biology*. Chapman and Hall, New York, 2006.
4. A. Amir, A. Apostolico, G.M. Landau, and G. Satta. Efficient text fingerprinting via parikh mapping. *J. Discrete Algorithm*, 1(5–6):409–421, 2003.

5. A. Amir, E. Chencinski, C. Iliopoulos, T. Kopelowitz, and H. Zhang. Property matching and weighted matching. *Theor Comput Sci*, pp. 298–310, 2006.
6. A. Amir, C. Iliopoulos, O. Kapah, and E. Porat. Approximate matching in weighted sequences. *Combin Pattern Matching*, pp. 365–376, 2006.
7. A. Andersson, N.J. Larsson, and K. Swanson. Suffix trees on words. *CPM '96: Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, London, UK, 1996, New York, pp. 102–115. Springer-Verlag.
8. P. Antoniou, M. Crochemore, C. Iliopoulos, I. Jayasekera, and G. Landau. Conservative string covering of indeterminate strings. *Proceedings of the 13th Prague Stringology Conference (PSC 2008)*, Prague, Czech Republik, 2008.
9. P. Antoniou, C. Iliopoulos, I. Jayasekera, and W. Rytter. Computing repetitive structures in indeterminate strings. *Proceedings of the 3rd IAPR, International Conference on Pattern Recognition in Bioinformatics (PRIB)*, Melborne, Australia, 2008.
10. A. Apostolico, C. Iliopoulos, G. Landau, B. Schieber, and U. Vishkin. Parallel construction of a suffix tree. *Algorithmica*, 3:347–365, 1988.
11. R. Baeza-Yates and G.H. Gonnet. A new approach to text searching. *Commun ACM*, 35(10):74–82, 1992.
12. S. Bedathur and J. Haritsa. Engineering a fast online persistent suffix tree construction. *Proceedings of the 20th International Conference on Data Engineering*, Boston, MA, 2004, pp. 720–720.
13. S. Bedathur and J. Haritsa. Search-optimized suffix-tree storage for biological applications. *IEEE International Conference on High Performance Computing*, 2005, pp. 29–39.
14. M.A. Bender, M. Colton, and B. Kuzsmaul. Cache-oblivious string b-trees. *25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS06)*, Chicago, IL, 2006, pp. 233–242.
15. G. Brodal and R. Fagerberg. Cache-oblivious string dictionaries. *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Miami, Fl, 2006, pp. 581–590.
16. A.L. Brown. Constructing genome scale suffix trees. In Yi-Ping Phoebe Chen, editor, *Second Asia-Pacific Bioinformatics Conference (APBC2004)*, volume 29 of *CRPIT*. ACS, Dunedin, New Zealand, 2004.
17. S. Burkhardt and J. Karkkainen. Fast lightweight suffix array construction and checking. In *Symposium on Combinatorial Pattern Matching*, Springer Verlag, LNCS, New York, 2003.
18. C. Cheung, J. Yu, and H. Lu. Constructing suffix tree for gigabyte sequences with megabyte memory. *IEEE Trans Knowl Data Eng*, 17(1):90–105, 2005.
19. M. Christodoulakis, C. Iliopoulos, L. Mouchard, K. Perdikuri, A. Tsakalidis, and K. Tsichlas. Computation of repetitions and regularities on biological weighted sequences. *J Comput Biol*, 13(6):1214–1231, 2006.
20. D. Clark and I. Munro. Efficient suffix trees on secondary storage. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, 1996, pp. 383–391.
21. R. Clifford and M. Sergot. Distributed and paged suffix trees for large genetic databases. *Combinatorial Pattern Matching, 14th Annual Symposium*, Michocan, Mexico, 2003, pp. 70–82.

22. R. Cole, M. Gottlieb, and L. Lewenstein. Dictionary matching and indexing with errors and dont cares. *Proceedings of the 36th Annual Symposium on Theory of Computing (STOC)*, Chicago, Il, 2004, pp. 91–100.
23. R. Cole, M. Kopelowitz, and L. Lewenstein. Suffix trays and suffix trists: structures for faster text indexing. *Proceedings of the 33rd International Colloquium on Automata Languages and Programming(ICALP)*, Venice, Italy, 2006, pp. 358–369.
24. A. Crauser and P. Ferragina. A theoretical and experimental study on the construction of suffix arrays in external memory. *Algorithmica*, 32:1–35, 2002.
25. R. Dementiev, J. Karkkainnen, J. Menhert, and P. Sanders. Better external memory suffix array construction. *Workshop on Algorithm Engineering and Experiments*, 2005, pp. 86–97.
26. D.K. Kim, J. Sim, H. Park, and K. Park. Linear time construction of suffix arrays. *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching*, Michoacan, Mexico, 2003, pp. 186–199.
27. M. Farach. Optimal suffix tree construction with large alphabets. *38th Annual Symposium on the Foundations of Computer Science (FOCS)*, New York, 1997, pp. 137–143.
28. P. Ferragina. String search in external memory: Data structures and algorithms. In S. Aluro, editor, *Handbook of Computational Molecular Biology*, Chapman & Hall, New York, 2006.
29. P. Ferragina, R. Gonzalez, G. Navarro, and R. Venturini. Compressed text indexes: From theory to practice. *ACM J Exper Algorithmics*, 13, 2008.
30. P. Ferragina and R. Grossi. The string b-tree: A new data structure for string search in external memory and its applications. *J ACM*, 46(2):236–280, 1999.
31. P. Ferragina, R. Grossi, A. Gupta, R. Shah, and J. Vitter. On searching compressed string collections cache-obliviously. *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Vancouver, BC, Canada, 2008, pp. 181–190.
32. P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness and beyond. *IEEE Symposium on Foundations of Computer Science*, pp. 184–196, 2005.
33. M.J. Fischer and M.S. Paterson. String-matching and other products. Technical Report, Cambridge, MA, 1974.
34. T. Flouri, C. Iliopoulos, M. Sohel Rahman, L. Vagner, and M. Voracek. Indexing factors in dna/rna sequences. *BIRD*, pp. 436–445, 2008.
35. G. Manzini. Two space saving tricks for linear time lcp array computation. In J. Gudmundssson, editor, *Scandinavian Workshop on Algorithm Theory*. Springer, New York, 2004.
36. G.H. Gonnet, R.A. Baeza-Yates, and T. Snider. New indices for text: Pat trees and pat arrays. In B. Frakes and R.A. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, Prentice Hall, Upper Saddle R, 1992.
37. G. Grillo, F. Licciuli, S. Liuni, E. Sbisa, and G. Pesole. Patsearch: A program for the detection of patterns and structural motifs in nucleotide sequences. *Nucleic Acids Res*, 31:3608–3612, 2003.
38. R. Grossi, A. Gupta, and J. Vitter. When indexing equals compression: Experiments on suffix arrays and trees. In *ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, LA, 2004, pp. 636–645.

39. R. Grossi and J.S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC)*, ACM Press, New York, 2000, pp. 397–406.
40. M. Gupta and J. Liu. Discovery of conserved sequence patterns using a stochastic dictionary model. *J Am Stat Assoc*, 98:55–66, 2003.
41. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997.
42. D. Gusfield and S. Orzack. Haplotype inference. In S. Aluru, editor, *Handbook of Computational Molecular Biology,* Chapman and Hall, New York, 2006.
43. R. Hariharan. Optimal parallel suffix tree construction. *J Comput Syst Sci*, 55(1):44–69, 1997.
44. J. Holub and W.F. Smyth. Algorithms on indeterminate strings. *Proeedings of the 14th Australasian Workshop on Combinatorial Algorithms*, Seoul, Korea, 2003.
45. J. Holub, W.F. Smyth, and Shu Wang. Fast pattern-matching on indeterminate strings. *J Discrete Algorithm*, 6:37–50, 2006.
46. W. Hon, T.W. Lam, K. Sadakane, and W. Sung. Constructing compressed suffix arrays with large alphabets. *International Symposium on Algorithms and Computation*, Kolkata, India, 2006, pp. 505–516.
47. W. Hon, K. Sadakane, and W. Sung. Breaking a time-space barrier in constructing full-text indices. *IEEE Symposium on Foundations of Computer Science*, 2003, pp. 251–260.
48. J. Hughes, P. Estep, S. Tavazoie, and G. Church. Computational identification of cis-regulatory elements associated with groups of functionally related genes in sacharomyces cerevisae. *J Mol Biol*, 296:1205–1214, 2000.
49. E. Hunt, M. Atkinson, and R. Irving. A database index to large biological sequences. *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers, San Francisco, CA, 2001.
50. C. Iliopoulos, I. Jayasekera, B. Melichar, and J. Supol. Weighted degenerated approximate pattern matching. *Proceedings of the 1st International Conference on Language and Automata Theory and Applications*, Tarragona, Spain, 2007.
51. C. Iliopoulos, M. Mohamed, L. Mouchard, W. Smyth, K. Perdikuri, and A. Tsakalidis. String regularities with don't cares. *Nordic J Comput*, 10(1):40–51, 2003.
52. C. Iliopoulos, M. Rahman, and W. Rytter. Algorithms for two versions of lcs problem for indeterminate strings. *International Workshop on Combinatorial Algorithms* (IWOCA), Newcastle, Australia, 2007.
53. C. Iliopoulos, M. Rahman, M. Voracek, and L. Vagner. Computing constrained longest common subsequence for degenerate strings using finite automata. *Proceedings of the 3rd Symposium on Algorithms and Complexity*, Rome, Italy, 2007.
54. C.S. Iliopoulos, C. Makris, Y. Panagis, K. Perdikuri, E. Theodoridis, and A. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundamenta Informaticae*, 71(2–3):259–277, 2006.
55. H. Itoh and H. Tanaka. An eficient method for in memory construction of suffix arrays. *Symposium on String Processing and Information Retrieval*, 2006, pp. 81–88.
56. R. Japp. The top-compressed suffix tree: A disk-resident index for large sequences. *Bioinformatics Workshop, 21st Annual British National Conference on Databases*, 2004.

57. L. Jensen and S. Knudsen. Automatic discovery of regulatory patterns in promoter regions based on whole cell expression data and functional annotation. *Bioinformatics*, 16:326–333, 2000.

58. J. Karkkainen and P. Sanders. Simple linear work suffix array construction. *Proceedings of 30th International Colloquium on Automata, Languages and Programming(ICALP)*, Eindhoven, The Netherlands, 2003, pp. 943–955.

59. T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-preffix computation in suffix arrays and its applications. *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*, Jerusalem, Israel, 2001, pp. 181–192.

60. S. Keles, M. van der Laan, S. Dudoit, B. Xing, and M. Eisen. Supervised detection of regulatory motifs in dna sequences. *Statistical Applications in Genetics and Molecular Biology*, 2, 2003.

61. D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading MA, 1998.

62. S. Kurtz. Reducing the space requirement of suffix trees. *Softw—Prac Exp*, 29(13):1149–1171, 1999.

63. G. Landau, D. Tsur, and O. Weimann. Indexing a dictionary for subset matching queries. *SPIRE*, pp. 195–204, 2007.

64. C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

65. H. Li, V. Rhodius, C. Gross, and E. Siggia. Identification of the binding sites of regulatory proteins in bacterial genomes. *Genetics*, 99:11772–11777, 2002.

66. Y. Liu, L. Wei, S. Batzoglou, D. Brutlag, J. Liu, and X. Liu. A suite of web-based programs to search for transcriptional regulatory motifs. *Nucleic Acids Res*, 32:204–207, 2004.

67. U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J Comput*, 25(5):935–948, 1993.

68. G. Manzini and P. Ferragina. Engineering a lightweight suffix array construction algorithm. *Algorithmica*, 40:33–50, 2004.

69. L. Marsan and M.F. Sagot. Extracting structured motifs using a suffix tree- algorithms and application to promoter concensus identification. *RECOMB*, pp. 210–219, 2000.

70. E.M. McCreight. A space-economical suffix tree construction algorithm. *J Assoc Compu Mach*, 23(2):262–272, 1976.

71. A. McGuire, J. Hughes, and G. Church. Conservation of DNA regulatory motifs and discovery of new motifs in microbial genomes. *Nucleic Acids Res*, 10:744–757, 2000.

72. D. Mehta and S. Sahni. *Handbook of Data Structures and Applications*. Chapman and Hall, New York, 2005.

73. J. Munro, V. Raman, and S.S. Rao. Space efficient suffix trees. *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer-Verlag, London, UK, 1998, pp. 186–196.

74. B. Phoophakdee and M. Zaki. Genome-scale disk-based suffix tree indexing. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, New York, 2007. ACM, pp. 833–844.

75. P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. *Combin Pattern Matching*, pp. 200–210, 2003.

76. M. Rahman, C. Iliopoulos, and L. Mouchard. Pattern matching in degenerate dna/rna algorithms. *Proceedings of the Workshop on Algorithms and Computation*, pp. 109-120, 2007.

77. K. Roh, M. Crochemore, C. Iliopoulos, and K. Park. External memory algorithms for string problems. *Fundamenta Informaticae*, 84(1):17–32, 2008.

78. D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Mach Learn*, 25:117–149, 1996.

79. D. Roth, J. Hughes, P. Esterp, and G. Church. Finding dna regulatory motifs within unaligned noncoding sequence clustered by whole genome mrna quantitation. *Nat Biotechnol*, 16:939–945, 1998.

80. K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *J Algorithms*, 48(2):294–313, 2003.

81. S. Sahinalp and U. Vishkin. Symmetry breaking for suffix tree construction. *26th Annual ACM Symposium on the Theory of Computing*, Quebec, Canada, 1994, pp. 300–309.

82. Z. Sun, J. Yang, and J. Deogun. Misae: A new approach for regulatory motif extraction. *Proceedings of the IEEE Computational Systems Bioinformatics Conference*, 2004.

83. M. Tadesse, M. Vannucci, and P. Lio. Identification of dna regulatory motifs using bayesian variable selection suite. *Bioinformatics*, 20:2553–2561, 2004.

84. S. Tata, R. Hankins, and J. Patel. Practical suffix tree construction. *30th International Conference on Very Large Data Bases*, Ontario, Canada, 2004, pp. 36–47.

85. W. Thompson and C. Lawrence. Gibbs recursive sampler: Finding transcription factor binding sites. *Nucleic Acids Res*, 31:3580–3585, 2003.

86. Y. Tian, S. Tata, R. Hankins, and J. Patel. Practical methods for constructing suffix trees. *VLDB J*, 14(3):281–299, 2005.

87. M. Tompa and S. Sinha. A statistical method for finding transcription factor binding sites. *Proceedings Internatioanl Conference on Intelligent Systems in Molecular Biology*, pp. 37–45, 2000.

88. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.

89. J.S. Vitter. *Algorithms and Data Structures for External Memory*. Publishers Inc., Hanover, USA, 2008.

90. M. Voracek, B. Melichar, and M. Christodoulakis. Generalized and weighted strings: Repetitions and pattern matching. In *String Algorithmics*, KCL Publications, King's College London, 2004, pp. 225–248.

91. M. Voracek, V. Vagner, and T. Flouri. Indexing degenerate strings. *Proceedings of International Conference on Computational Methods in Science and Engineering*. American Mathematical Institute of Physics, 2007.

92. P. Weiner. Linear pattern matching algorithms. In *14th IEEE Annual Symposium on Switching and Automata Theory*, 1973, pp. 1–11.

93. S. Wu and U. Manber. Fast text searching: Allowing errors. *Commun ACM*, 35(10):83–91, 1992.

# 2

# EFFICIENT RESTRICTED-CASE ALGORITHMS FOR PROBLEMS IN COMPUTATIONAL BIOLOGY

Patricia A. Evans and H. Todd Wareham

## 2.1 THE NEED FOR SPECIAL CASES

Many problems of interest, in computational biology as in other fields, have been proven to be NP-hard and so cannot be solved efficiently in general unless $P = \mathrm{NP}$ (the set of problems that can be solved nondeterministically in polynomial time). The large sizes and increasingly massive quantities of data make problem tractability and algorithm efficiency a critical concern for computational biology, and indeed, even polynomial-time algorithms can have difficulty coping with the large amounts of data typically encountered, necessitating the development of specialized algorithms to deal with these situations and applications.

Although intractability results are certainly a formidable obstacle to solving such problems and tend to lead researchers to use heuristics and approximations, the general form of each problem that has been proven hard rarely resembles the problem as it is applied. Reduction gadgets and constructions often describe data that does not look like the relevant biological data, leaving open the possibility that special cases that more closely resemble the intended application may be tractable.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

One key aspect that may lead to tractable special cases is that the data in biological problems often have characteristics that are small or limited in size. Lengths in particular can be relatively short, such as the length of a substring representing a motif and the length of a newly sequenced genome snippet. For genome data, the alphabet is also very small, consisting of four bases (possibly with the addition of wildcards or limited unknowns).

For cases in which one or more characteristics of the data are small, algorithms for the tractable special cases can be identified using techniques from the theory of parameterized complexity [15]. Parameterized complexity examines the tractability of a problem relative to one or more parameter, characteristics of the problem that potentially can be restricted to small values to provide an efficient algorithm for those cases. Such fixed-parameter tractable algorithms can provide practical and accurate results for otherwise intractable problems, and there is an extensive and expanding set of techniques for designing and improving fixed-parameter algorithms [15, 42].

Intractability in the parameterized setting is determined by showing hardness for one of the classes in the parameterized complexity $W$-hierarchy. Because the hardness or tractability can be affected by the choice of parameters, analyzing the results for the same problem with different parameters leads to insights about the effect of these parameters on the difficulty of the problem and ultimately defines which parameter-based special cases and related applications are tractable.

Not all special cases are defined by parameters. To make the problems tractable, properties of input sequences and structures often need to be restricted to cases that resemble biological data, which will change naturally depending on the application being examined. In this chapter, we examine the relevant parameters and special cases for several sequence and string problems applicable to computational biology problems, namely Shortest Common Superstring (SCS), Longest Common Subsequence (LCS), and Common Approximate Substring (CAS). We present the different known tractable and intractable variants of these problems, showing which restrictions lead to usable algorithms and also define those variants for which further research is needed.

## 2.2 ASSESSING EFFICIENT SOLVABILITY OPTIONS FOR GENERAL PROBLEMS AND SPECIAL CASES

Two basic questions must be addressed by a formal efficiency analysis of a computational problem:

1. Can the problem as given be solved efficiently?
2. If not, can particular restrictions of that problem be solved efficiently?

In this section, we will outline briefly how these questions can be answered using techniques from the classical and parameterized theories of computational complexity [15, 23].

In regards to the first question, we will adopt the common notion of efficiency in computer science—namely, we will say that a computational problem is (polynomial-time) tractable if it can be solved by an an algorithm whose worst-case running time is bounded by some function $n^c$, where $n$ is the input size and $c$ is a constant.[1] If no such algorithm exists, then the problem is (polynomial-time) intractable. Super-polynomial time algorithms generally are considered impractical because they have unrealistically long running times for all but small inputs.

We can establish polynomial-time tractability by simply giving an algorithm for a problem that runs in polynomial time. Establishing polynomial-time intractability typically is done by proving that the problem is at least as computationally difficult as every problem in a class $X$ of known intractable problems (*i.e.*, the problem is X-hard). For details of how this is done, the interested reader is referred to [23].[2]

If our problem of interest is polynomial-time intractable, then how might we show that it is tractable (in some sense) under a particular restriction? There are two possible types of restrictions:

1. Structural restrictions (*i.e.*, restrictions phrased in terms of structural properties of the inputs given or outputs requested in the problem)
2. Parameterized restrictions (*i.e.*, restrictions phrased in terms of the numerical values of one or more aspects of the problem)

Structural restrictions can be addressed by defining new versions of the problem that incorporate these restrictions and by assessing the polynomial-time tractability of these new problems as described by. Parameterized restrictions require a new way of thinking. As our problem is polynomial-time intractable, all algorithms solving that problem run in super-polynomial time; however, if this time is super-polynomial only in the aspects being restricted (whose values are assumed to be very small) and polynomial in all other aspects of input size, then the resulting running time may in practice be effectively polynomial and hence reasonable. Let us call such a set $p$ of one or more simultaneously restricted aspects of a problem $\Pi$ a parameter of $\Pi$, and denote the version of $\Pi$ restricted relative to $p$ by $\langle p \rangle$-$\Pi$.

This looser conception of tractability under parameterized restrictions is captured by the following definition:

> **Definitio** : A problem $\Pi$ is fixed-parameter tractable (fp-tractable) relative to a particular parameter $p$ if $\Pi$ is solvable by an algorithm with running time bounded by $f(p)n^c$, where $f$ is an arbitrary function, $n$ is the input size, and $c$ is a constant.

[1] Such running times often are stated in terms of $O()$-notation, where $O(g(n))$ is the set of functions $f(n)$ that are asymptotically upperbounded by $g(n)$ (*i.e.*, $f(n) \leq c \times g(n)$) for all $n \geq n_0$ for some constants $c$ and $n_0$ (*cf.* Footnote 4).

[2] Ideally, we want to show $X$-hardness relative to an $X$ such that $P \subset X$ (*i.e.*, $X$ properly contains the class $P$ of polynomial-time solvable problems). However, we often only can show hardness for an $X$ such that $P \subseteq X$, and we have strong empirical support (though not mathematical certainty) that $P \neq X$. This is the case in this chapter in which all our polynomial-time intractability results are demonstrated by *NP*-hardness, which is acceptable, as the conjecture $P \neq NP$ has very strong empirical support; again, the interested reader is referred to [23] for details.

If no such algorithm exists for $\Pi$ relative to $p$, then $\Pi$ is said to be fp-intractable relative to $p$ (or, equivalently, $\langle p \rangle$-$\Pi$ is fp-intractable). Note that as a problem may be fp-tractable relative to some parameters and fp-intractable relative to others, fp-(in)tractability always must be stated relative to a parameter.

This is the conception of tractability underlying the theory of parameterized computational complexity created by Downey and Fellows [15]. As in the case of polynomial-time tractability, we show fp-tractability simply by giving an algorithm for the problem with the required running time relative to the specified parameter (often by invoking specialized techniques [42]), and we show fp-intractability of a problem relative to a specified parameter by showing the problem-parameter combination is hard relative to a class in the W-hierarchy = $\{W[1], W[2], \ldots, W[P], \ldots XP\}$.[3] Again, as the details of how this is done need not concern us here, the interested reader is referred to [15]. The class of fp-tractable problems is FPT.

In analyses of a problem's complexity, intractability proofs (by virtue of their generality and intricacy) often take center stage and are given the most attention. However, it is worth remembering that our ultimate goal is to solve efficiently the given problem or a useful restricted version of this problem. Given this, intractability results assume their proper role—namely, delimiting which versions of a problem *cannot* be solved efficiently, hence, both highlighting and allowing us to focus more productively our energies on developing the best possible algorithms for versions of the problem that *can* be solved efficiently.

## 2.3 STRING AND SEQUENCE PROBLEMS

Three central string-based problems in computational biology are:

1. **Sequence Reconstruction**: Given a set of sequence-fragments, we reconstruct the original sequence.
2. **Sequence Alignment**: Given a set of sequences, we derive the best overall global alignment of these sequence to highlight both corresponding and divergent elements of these sequences.
3. **Sequence Consensus**: Given a set of sequences, we derive the best consensus sequence, summarizing corresponding and divergent elements of these sequences.

As genomic regions of interest range in size from several thousand (individual genes) to millions or billions (whole genomes) of nucleotides in length and because current technologies only can sequence regions less than 2000 nucleotides long reliably [47], sequence reconstruction is a critical first step in any sequence-level genomic analysis.

[3] Analogous to Footnote 2, using such hardness results to establish fp-intractability is acceptable as the conjecture $FPT \neq W[1]$ has strong empirical support, where $FPT$ is the class of fp-tractable problem-parameter combinations. The interested reader is referred to [15] for details.

Subsequent analysis of groups of two or more sequences often are based on sequence alignments. For example, as random mutations that occur in functionally significant regions of sequences are typically deleterious and thus will not be passed on to future generations, highly similar regions in an alignment of sequences that have evolved from a common ancestral sequence frequently are assumed to be of functional significance and thus can be used to unravel protein and regulatory functions in the cases of coding and noncoding regions, respectively. Consensus sequences, in addition to being concise summaries of corresponding regions in alignments, have their own applications. For example, under certain notions of consensus, consensus sequences specify short complementary strings that can bind to a specific region in each of a given set of sequence, and are therefore potential universal primers for polymerase chain reaction sequencing reactions or drug targets.

In the following sections, we will give overviews of both general and special-case algorithms for the formal computational problems associated with each of these problems—namely, Shortest Common Superstring (Section 2.4), Longest Common Subsequence (Section 2.5), and Common Approximate Substring (Section 2.6).

## 2.4 SHORTEST COMMON SUPERSTRING

The most basic formal computational problem associated with sequence reconstruction is the following:

Shortest Common Superstring (SCSt)
*Input*: A set $S = \{s_1, s_2, \ldots, s_k\}$ of strings over an alphabet $|\Sigma|$.
*Output*: The shortest string $s'$ such that each string $s \in S$ is a substring of $s'$.

This problem is an idealization of actual sequencing under currently available technologies on several counts:

1. The DNA strand from which fragments originated typically is not known, and fragments thus may be complementary and reversed relative to the coding strand.
2. Errors may occur in determining the sequence of any fragment.
3. Depending on the genomic region being sequenced, repetitive sequence regions may be collapsed together and hence not reconstructed correctly in any shortest common superstring.

The first difficulty can be accommodated by requiring for each $s \in S$ that either $s$ *or* rev(comp($s$)) be a substring of $s'$, where rev($s$) returns the reversed version of string $s$ and comp($s$) returns the base-complemented version of DNA or RNA string $s$ (*i.e.*, rev(*ATTC*) = *CTTA*, comp(*ATTC*) = *TAAG*, and comp(*AUUC*) = *UAAG*.) The second difficulty can be accommodated by requiring that each $s \in S$ match some substring $s''$ of $s'$ such that dst($s, s''$) $\leq \epsilon$, where dst() is a distance function on pairs of strings measuring the degree of error required to produce $s$ from $s''$ and $\epsilon$ is an acceptable sequencing-error threshold. Denote the version of SCSt incorporating these modifications by SCSt-re. The third difficulty is much more problematic, as it is a product of the parsimony assumption underlying the requirement that produced superstrings be the shortest possible. However, this difficulty, to a large extent, can be eliminated by either including fragments in $S$ that are long enough to span the regions between repeats, or incorporating extra information about the ordering of fragments on the genome; as such techniques are beyond the scope of this chapter, the interested reader is referred to [48] and references for details.

In typical instances of DNA sequencing, the sequence-alphabet $\Sigma = \{A, G, C, T\}$ is of a small fixed size, the number of fragments $k$ can be on the order of thousands to millions, and the maximum fragment-length (denoted by $n = \max_{s \in S} |s|$) varies with the sequencing technology from fixed constants less than 10 to roughly 1000. This suggests that algorithms that are efficient relative to restricted alphabet and/or fragment size would be of use. Known intractability results suggest that polynomial-time efficiency in these cases is probably not possible, that is,

- SCSt is NP-hard when $n = 3$ or $|\Sigma| \geq 2$ [22].
- SCSt-re is NP-hard when $|\Sigma| = 4$ or $n = 15$ [54, Theorem 7.2].

The intractability of SCSt holds even if (1) $|\Sigma| = 2$, and one of these symbols occurs only three times in each $s \in S$ or (2) $n = 3$ and each $\sigma \in \Sigma$ occurs at most eight times in the strings in $S$ [36, 37] (see also [51]). Though it may be tempting to consider solution-superstrings whose lengths are within a small multiplicative factor of the length of the shortest superstring, it is known that such approximate superstrings cannot be derived in polynomial time to an arbitrary degree of accuracy even for $|\Sigma| = 2$ [8, 44, 52], and the best known polynomial-time approximation algorithm only can guarantee solutions whose lengths are less than or equal to $2.5 \times$ optimal [50] (see also [30]), which is not practical. However, as we will discuss, there may yet be acceptable exact algorithms for special cases.

In the remainder of this section, we will look at algorithms for solving the general shortest common superstring problem (Section 2.4.1) and the special case in which $|\Sigma|$ and $n$ are bounded simultaneously (Section 2.4.2).

### 2.4.1 Solving the General Problem

Courtesy of the NP-hardness results described, all exact algorithms for SCSt must run in exponential time. There are two general strategies for such algorithms:

1. Enumerate all possible solution superstrings and check for each superstring if it includes every $s \in S$ as a substring; return the shortest such common superstring.
2. Enumerate all possible solution superstrings generated by orderings of strings in $S$ that allow these strings to overlap; return the shortest such superstring.

Naive algorithms implementing these strategies have the following resource requirements:

- The longest possible solution superstring is simply a concatenation of all strings in $S$ and hence has length at most $k \times n$; thus, there are $\sum_{1=n}^{kn} |\Sigma|^i \leq (kn - (n-1))|\Sigma|^{kn} = O(kn|\Sigma|^{kn})$ possible solution superstrings. As the inclusion of a string $x$ as a substring in string $y$ can be checked in $O(|x| + |y|)$ using a suffix-tree based algorithm [25, Section 7.1], the first strategy runs in $O(kn|\Sigma|^{kn} \times k(kn+n)) = O(|\Sigma|^{kn}k^3n^2)$ time and $O(kn)$ space.
- The number of possible orderings of the strings in $S$ is $k! = O(k^k)$. In any shortest superstring based on such an ordering, a pair of adjacent strings in this ordering will have the maximum overlap possible (otherwise, the maximum overlap potentially could be used to create a shorter superstring, which is a contradiction). As the maximum overlap between two strings from $S$ can be computed in $O(n)$ time, the second strategy runs in $O(k^kn)$ time and $O(kn)$ space.

The actual (though not worst-case) run time of the second strategy can be improved by exploiting the incremental manner in which solution superstrings are created in this strategy. For example, a branch-and-bound algorithm such as that in [6] could evaluate all orderings in a search tree in which each level adds the next string in the generated ordering. In such a search tree, nodes generating superstrings longer than the best seen so far can be pruned, potentially eliminating a large proportion of orderings of $S$ from even being considered. Alternatively, orderings could be encoded implicitly in a directed edge-weighted complete graph whose vertices correspond to the strings in $S$, and arcs $(s_i, s_j)$, $1 \leq i, j, \leq k$, have weight equal to the maximum overlap of $s_i$ with $s_j$ (which may be 0 if the strings do not overlap). Given such an overlap graph, the shortest superstring can be derived by finding the maximum-weight Hamiltonian path in this graph. Though this overlap graph algorithm for SCSt is elegant, it requires more (specifically, $O(k^2n)$) space; moreover, the running time is still exponential, as the problem of finding maximum-weighted Hamiltonian paths is NP-hard [23, Problem GT39].

Both of these strategies can be adapted (albeit at increased computational cost) to handle the fragment reversal and sequencing errors difficulties associated with actual sequencing. In the case of the first strategy, for each $s \in S$, both $s$ and rev(comp($s$)) can be checked against the solution superstring using the error-measure string-comparison function dst(). Assuming a sequencing-error model allowing base substitutions, insertions, and deletions, dst() is pairwise edit distance, which can be computed in in $O(|x||y|)$ time and $O(|x| + |y|)$ space [25, Section 11]. The run-time and space requirement increase is more dramatic in the case of the second strategy; not only is the number of orderings of $S$ increased by a factor of $2^k$ (each string $s$ in the ordering is now either $s$ or rev(comp($s$))), but pairs of adjacent strings in the ordering can overlap in more than one way (as we must allow errors). Further increases come from allowing errors in regions of strings that in $s$ that do not overlap with other strings, as well as coordinating errors when more than two strings overlap in the solution superstring.

In the case of the second strategy, it is possible to handle repeats using the approach proposed by Myers [40] in which an overlap graph is processed to create a string graph in which each arc is either required (can be traversed at most once), exact (must be traversed exactly once), or optional (can be traversed any number of times). In such a string-graph, optional arcs allow fragments to be included more than once in a solution superstring corresponding to a minimum-length walk that respects all arc-constraints, hence, allowing repeats to be reconstructed. Though the processing to create string graphs from overlap graphs can be done efficiently [40], any algorithm implementing this approach is still exponential time because the problem of finding a minimum-length constraint-respecting walk in a string-graph is NP-hard [35, Theorem 1].

### 2.4.2 Special Case: SCSt for Short Strings Over Small Alphabets

Interest in this special case first developed with the development of various technologies in the mid-1980s for rapidly assessing which subset $S$ of the strings in a set $C$ of length-$n$ DNA strings ($n$-mers) are present in a given DNA string (*e.g.*, oligonucleotide arrays and DNA chips). The hope was that given this information, it would be possible to determine the sequences of short DNA strands (so-called sequencing by hybridization (SBH)) faster than using conventional Sanger-based technologies.

There is, in fact, a linear-time algorithm for ideal SBH in which no $n$-mer in $S$ occurs more than once in the sequence $s'$ to be reconstructed, and all $n$-mers in $s'$ have been read correctly (*i.e.*, $S$ is complete). This algorithm relies on a variant of overlap graphs called de Bruijn graphs. In a de Bruijn graph, it is the arcs rather than the vertices that correspond to the $n$-mers in $S$ and the vertices are the set of $(n-1)$-mers that occur in the strings in $S$. In particular, there is an arc between vertices $x$ and $y$ in a de Bruijn graph if there is an $n$-mer $z$ in $S$ such that $x$ is the prefix $(n-1)$-mer of $z$ and $y$ is the suffix $(n-1)$-mer of $z$. As $S$ is complete and all $n$-mers in $S$ occur exactly once in $s'$, the sequence of $s'$ can be reconstructed from any path in the de Bruijn graph that uses each arc exactly once (*i.e.*, an Euler path). Unlike the computation of Hamiltonian paths through all vertices in an overlap graph, which is NP-hard, the computation of Euler paths can be done in time linear in the size of the graph. Hence, as the de Bruijn graph corresponding to a given set $S$ can be constructed in time linear in the size of $S$, the ideal SBH algorithm runs in linear time.

Early theoretical analyses of the occurrences of repeats in random DNA strings suggested that sets $C$ composed of complete sets of $n$-mers could be used to reconstruct sequences with lengths up to $\sqrt{2 \times 4^n}$ [1,16]). However, it has been difficult to achieve this level of success because actual DNA sequence has statistical irregularities even in relatively short regions, and it has proven to be much more difficult than expected for all $n$-mers in a given sequence to be detected reliably on DNA chips, because of $n$-mer probe cross-hybridization and hybridization signal misreading.

The net effect of these problems is that the produced $S$ not only may contain more than one copy of an $n$-mer (*i.e.*, $S$ is a multiset), but that there may also be $n$-mers in $S$ that are not in $s'$ (positive errors) and $n$-mers in $s'$ that are not in $S$ (negative errors). As we can no longer guarantee that all and only elements of $s'$ are present in $S$, reconstructing $s'$ as a shortest superstring of the strings in $S$ is unacceptable. The more reasonable approach, advocated by Błazewicz *et al.* [6,7] is rather to look for a superstring of length less than a specified threshold $l$ that has the maximum possible number of elements of $S$ as substrings (note that the threshold is required as different quantities simultaneously cannot be optimized in a problem). Błazewicz *et al.* [6] give search-tree-based algorithms (analogous to the search-tree algorithm for SCSt described in Section 2.4.1) for both this variant of SCSt and the variant of SCSt incorporating only positive errors. These algorithms run in $O(k^k nl)$ time and $O(\min(kn, l))$ space but perform much faster in practice (particularly so if only positive errors are present). It is unlikely that algorithms with subexponential running times will be found, as Błazewicz and Kasprzak subsequently have shown that both of these variants of SCSt (as well as the variant incorporating only negative errors) are NP-hard [7, Theorems 1 and 2].

A hybrid overlap-de Bruijn graph approach to dealing with the presence of $n$-mer repeats in given sequence-fragments was proposed by Pevzner *et al.* [46]. In this approach, conventional arbitrary-length sequence fragments are used to create de Bruijn graphs relative to a specified length $n$ by decomposing each sequence-fragment into its associated set of overlapping $n$-mers. The sequence then is reconstructed by finding a minimal superwalk in the de Bruijn graph that includes the walks corresponding to each given sequence-fragment (note that these are walks instead of paths because individual sequence-fragments may contain $n$-mer repeats). No exact algorithm for solving this problem has yet been given in the literature. However, it is unlikely that a nonexponential time-exact algorithm exists, as the problem of finding minimal superwalks in de Bruijn graphs has been shown to be NP-hard for $|\Sigma| \geq 3$ and any $n \geq 2$ [35, Theorem 2].

### 2.4.3 Discussion

Table 2.1 summarizes known parameterized results for Shortest Common Superstring, considering the number of fragments and the fragment length as potential parameters together with different possible restrictions on the alphabet size. Though some variants are fp-tractable, the running times of the best known algorithms for these variants are still prohibitive in practice. Hence, all currently used assemblers are based on heuristics [48].

**Table 2.1 The parameterized complexity of Shortest Common Superstring**

| | Alphabet Size $\|\Sigma\|$ | | |
|---|---|---|---|
| Parameter | Unbounded | Parameter | Constant |
| – | NP-hard | $\notin XP$ | NP-hard |
| $k$ | FPT | FPT | FPT |
| $n$ | $\notin XP$ | $\notin XP$ | NP-hard |

Given the existence of the parameterized algorithms discussed, especially for multiparameter combinations such as $\langle|\Sigma|, k\rangle$-SCSt, further work needs to be done to find faster and more useful algorithms within these regions of tractability. Future research needs to focus on the special cases most suitable for sequence assembly problems, especially with the different characteristics (most notably the short read length together with constant size alphabet, still NP-hard) produced by recent advances in next-generation sequencing technology [48]. Additional special cases that incorporate errors, error rates, and how repeats are handled are also worthy of investigation to find algorithms tailored to the current input data characteristics.

## 2.5 LONGEST COMMON SUBSEQUENCE

The most basic formal computational problem associated with sequence alignment is the following:

Longest Common Subsequence (LCS)
*Input*: A set $S = \{s_1, s_2, \ldots, s_k\}$ of strings over an alphabet $|\Sigma|$
*Output*: The longest string $s'$ such that $s'$ is a subsequence of each string $s \in S$

This problem is an idealization of sequence alignment in that LCS contains all and only exactly corresponding symbols in the given sequences in $S$ and does not indicate explicitly how symbols that do not match exactly can correspond. Hence, LCS is a restricted case of the general sequence alignment problem in which *any* function may be used to evaluate the costs of aligning various symbol positions across the sequences in $S$ [45, Section 3]. As LCS also summarizes all and only the exactly corresponding elements in the given sequences in $S$, LCS is a restricted case of the general sequence consensus problem [14, Section 3]. Algorithms for LCS are used occasionally directly for finding alignments and consensus sequences, [4, 43]; therefore, such algorithms and resource-usage lower bounds for LCS are also useful to the extent that they apply to the various restricted sequence alignment and consensus methods used in practice (*e.g.*, sum-of-pairs (SP) alignment, tree alignment see [25, section 14] and references).

In typical instances of DNA sequence alignment, the sequence-alphabet $\Sigma = \{A, G, C, T\}$ is of small fixed size; the number of sequences $k$ to be aligned can vary from two to several hundred, and the maximum sequence-length (denoted by $n = \max_{s \in S} |s|$) varies from several hundred to several million. Various situations also require a variant of LCS in which the requested length of the derived common subsequence is specified as part of the input; let us call this length $l$. This suggests that algorithms that are efficient relative to restrictions on any of these parameters would be of use. In the important case of pairwise alignment (*i.e.*, $k = 2$) many efficient quadratic time and space algorithms are known for both sequence alignment and LCS [5, 25]. However, when $k \gg 2$, known intractability results suggest that under many restrictions, polynomial-time efficiency is probably not possible, that is,

- LCS is NP-hard when $|\Sigma| \geq 2$ [33].
- $\langle k \rangle$-LCS is $W[t]$-hard for $t \geq 1$ [10, Theorem 2].
- $\langle l \rangle$-LCS is $W[2]$-hard [10, Theorem 3].
- $\langle k, l \rangle$-LCS is $W[1]$-complete [10, Theorem 1].
- $\langle k, |\Sigma| \rangle$-LCS is $W[t]$-hard for $t \geq 1$ [9].
- $\langle k \rangle$-LCS is $W[1]$-hard when $|\Sigma| \geq 2$ [47, Theorem 2].

Though it may be tempting to consider solution-subsequences whose lengths are within a small multiplicative factor of the length of the longest subsequence, it is known that such approximate subsequences cannot be derived in polynomial time within any constant degree of accuracy unless $P = \text{NP}$ [29]. However, as we will discuss, there may yet be acceptable exact algorithms for special cases.

In the remainder of this section, we will look at algorithms for solving the general longest common subsequence problem (Section 2.5.1) and the special cases in which the given sequences are very similar (Section 2.5.2) or in which each symbol in $|\Sigma|$ occurs at most a constant number of times in each $s \in S$ (Section 2.5.3).

### 2.5.1 Solving the General Problem

Courtesy of the NP-hardness result described, all exact algorithms for LCS must run in exponential time. There are two general strategies for such algorithms:

1. Enumerate all possible strings of length $m = \min_{s \in S} |s|$ (or, if it is given, $l$) and check if each such string is a subsequence of every string in $S$; return the longest such common subsequence.
2. Enumerate all possible ways in which individual symbol positions can be matched exactly over all strings in $S$ to generate common subsequences; return the longest such common subsequence.

Given that the longest common subsequence of two strings $x$ and $y$ can be computed in $O(|x||y|)$ time and space [5, 25], the naive algorithm implementing the first strategy runs in $O(|\Sigma|^m nm) = O(|\Sigma|^n n^2)$ $(O(|\Sigma|^l nl))$ time and $O(n^2)$ $(O(nl))$ space. Algorithms implementing the second strategy depend on the data structures used to store all possible matching generated subsequences for $S$. The two most popular alternatives based on either dynamic programming tables or edit graphs are described.

The second strategy typically is implemented as a dynamic programming algorithm that encodes all possible matching generated subsequences in a $k$-dimensional table $T$ with $\prod_{s \in S} |s| = O(n^k)$ entries. Each dimension of $T$ corresponds to one of the strings $s \in S$ and has range 0 to $|s|$ and entry $T[i_1, i_2, \ldots, i_k]$ contains the length of the longest common subsequence of the strings up to these indices (*i.e.*, $s_1[0..i_1], s_2[0..i_2], \ldots, s_k[0..i_k]$), where $s_x[0..i_y]$ is the substring of $s_x$ consisting of the first $i_y$ symbols of $s_x$. The values of each entry is specified by the recurrence

$$T[i_1, i_2, \ldots, i_k] = \begin{cases} 0 & \text{if any } \mathrm{i_j} = 0,\ 1 \leq \mathrm{j} \leq \mathrm{k} \\ T[i_1 - 1, i_2 - 1, \ldots, i_k - 1] + 1 & \text{if } \mathrm{s_1[i_1] = s_2[i_2]} = \ldots \\ & \quad = s_k[i_k] \\ \max_{c \in C(i_1, i_2, \ldots, i_k)} T[c] & \text{otherwise} \end{cases}$$

where $C(i_1, i_2, \ldots, i_k)$ is the set of $k$ entry coordinate vectors generated by subtracting one from each of the coordinate values in $(i_1, i_2, \ldots, i_k)$ in turn. Note that each time the second clause in the recurrence is invoked, a symbol match that is potentially part of a longest common subsequence is established across all $s \in S$. By applying this recurrence in a bottom-up manner, the table entries are filled in until the value of entry $T[|s_1|, |s_2|, \ldots |s_k|]$, the length of the longest common subsequence of $S$, is computed at which point a traceback procedure is used to reconstruct a (possibly nonunique) path of recurrence applications from $T[0, 0, \ldots, 0]$ to $T[|s_1|, |s_2|, \ldots, |s_k|]$ corresponding to a longest common subsequence of the strings in $S$. As most $k + 1$ table entries must be consulted in the process of filling in a table entry or reconstructing a path backward one step from a table entry, this algorithm runs in $O(kn^k + k^2n)$ time and $O(n^k)$ space.

The second strategy also can be implemented as a path-finding algorithm relative to an edit graph. An edit graph is essentially a directed acyclic graph corresponding to the dynamic programming table described, such that there is a vertex for each entry in the table, and there is an arc $(x = T[i_1, i_2, \ldots i_k], y = T[j_1, j_2, \ldots, j_k])$ if (1) $i_h = j_h - 1$ for $1 \leq h \leq k$ and $s_1[j_1] = s_2[j_2] = \ldots = s_k[j_k]$ or (2) $(i_1, i_2, \ldots, i_k) \in C(j_1, j_2, \ldots, j_k)$. As these two types of arcs correspond to the second and third clauses, respectively, in the recurrence discriber, a straightforward weighting scheme would be to assign arcs of the two types weights 1 and 0, respectively. Under this scheme, a maximum-weight directed path in the edit graph with weight $D$ between the vertices corresponding to $T[0, 0, \ldots, 0]$ and $T[|s_1|, |s_2|, \ldots, |s_k|]$ corresponds to a longest common subsequence with length $l = D$ (as each unit of weight corresponds to a symbol position that is matched across all strings in $S$). Though such paths can be computed in polynomial time, the weighting-scheme often is reversed (*i.e.*, the two types arcs are assigned weights 0 and 1, respectively, to take advantage of faster shortest-path algorithms). Under this scheme, the analogous shorted path of weight $D$ corresponds to a longest common subsequence with length $l = ((\sum_{s \in S} |s|) - D)/k$ (as each unit of weight corresponds to symbol that must be deleted in some string in $S$ such that all strings in $S$ can be converted to the longest common subsequence) [3, p. 328]. The running time and space of edit graph-based algorithms is slightly larger than that required by the dynamic programming algorithm; however, as we will see below in Section 2.5.2, edit graphs have properties that can be exploited when solving certain special cases of LCS.

### 2.5.2 Special Case: LCS of Similar Sequences

This case is of interest when sequences that are very closely related (and hence very similar) are compared. The simplest way to exploit such similarity is to assume that the sequences are within a specified distance $b$ of each other (*i.e.*, at most, $b$ deletions are required in the given strings to convert them into the longest common subsequence). If $b$ is known in advance, then observe that the path in the dynamic programming table corresponding to the longest common subsequence only passes through entries within a $b$-width "band" surrounding the hypothetical diagonal extending from $(0, 0, \ldots, 0)$ that indicates all perfect matches across all sequences (this is because each deletion can cause the path to diverge at most one unit outward from this hypothesized 0-diagonal). Hence, it suffices to construct and operate on only this part of the table, which consists of $O(bn^{k-1})$ cells, reducing both time and space required for the overall dynamic programming algorithm by an order of magnitude. This algorithm, sketched in [27, Section 4], is a generalization of the band approach to aligning a pair of sequences described in [12].

General LCS algorithms dynamically minimize the portion of the dynamic programming table or edit a graph that is explored in response to similarity in the given sequences and, hence, do not require that distance-threshold $b$ be specified as input—namely, the first ("lazy dynamic programming") algorithm given in [28] and the shortest-path edit graph-based algorithm in [3]. Both algorithms are generalizations of the algorithms in [38,55], which essentially greedily construct a path in a dynamic programming table or edit graph by starting at $T[0, 0, \ldots, 0]$ on the 0-diagonal and iteratively traveling as far as possible along the current diagonal before skipping to and resetting the current diagonal to the most promising (according to a distance-estimate) of the closest diagonals until $T[|s_1|, |s_2|, \ldots, |s_k|]$ is reached. The algorithm in [28] runs in $O(kn(n-l)^{k-1})$ time and space and the algorithm in [3] runs in $O(nD^{k-1})$ time and space (though the space can be reduced to $O(kn + nD^{k-1})$ at the cost of doubling the runtime [3, Section 4]).

The theoretical runtime savings of both these algorithms improves dramatically as the similarity of the strings in $S$ increases; however, there may be constant factors hidden by the asymptotic $O$-notation that boost actual run times. Experiments reported in Barsky *et al.* [3] suggest that their algorithm has low run times even for moderately similar strings, outperforming the general dynamic programming algorithm for LCS described in Section 2.5.1 even when strings are as little as 50% similar (*i.e.*, $l/n = 0.5$ (*cf.* experiments reported in [28] which show their algorithm only outperforms at 90% similarity or above)). That being said, it is important to note that in the worst case in which the strings have no symbols in common and there is no common subsequence (*i.e.*, $l = 0$ and $D = k$), both these algorithms have time complexities that are comparable with or even slightly worse than the general dynamic programming algorithm for LCS.

### 2.5.3 Special Case: LCS Under Symbol-Occurrence Restrictions

This case is of interest when the strings being modeled are orders of homologous genes on chromosomes in different organisms in which each organism has a small number of copies (often one) of each type of gene; thus, $|\Sigma|$ can be as large as $n$ or even $O(kn)$ if there are genes unique to particular organisms (*cf.* $|\Sigma| = 4$ for DNA sequences). Alignment of such gene-order sequences is useful in gene prediction and certain genomic-level variants of evolutionary tree reconstruction (see [39] and references).

Such gene-order sequences can be modeled by $p$-sequences [20]. A $p$-sequence is a string $s$ over an alphabet $\Sigma$ in which each symbol in $\Sigma|$ occurs at most once; if every symbol in $\Sigma$ occurs exactly once (*i.e.*, $s$ is a permutation of $\Sigma$), then $s$ is a complete $p$-sequence. Let $p$-LCS denote the special case of LCS in which all $s \in S$ are $p$-sequences. When all $s \in S$ are complete $p$-sequences, $p$-LCS can be solved in in $O(kn(k + \log n))$ time [20, theorem 6].

It turns out that this polynomial-time solvability still holds for general $p$-sequences and small sets of sequences in which each symbol in $\Sigma$ is allowed to occur at most some constant $c > 1$ times. Let us call a sequence in which each symbol in $\Sigma$ occurs at most $o$ times a $p(o)$-sequence, and let $p(o)$-LCS denote the variant of LCS that operates over $p(o)$-sequences for an $o$ specified in the input; note that $p(1)$-LCS is equivalent to $p$-LCS when $o = 1$ and to LCS when $o = n$.

To show these results, we can use any one of several LCS algorithms whose run times are low when the number of occurrences of each symbol of $\Sigma$ in each string of $S$ is small [2,26]. These algorithms restrict the portions of the dynamic programming table that they explore by focusing on match points. A match point of a set $S$ of $k$ strings is a vector $(i_1, i_2, \ldots, i_k)$ such that $s_1[i_1] = s_2[i_2] = \ldots = s_k[i_k]$ (*i.e.*, entries in the dynamic programming table whose entries are filled in using the second clause of the LCS recurrence given in Section 2.5.1). Note that match points correspond to possible elements in a longest common subsequence. The algorithms in [26] and [2] essentially encode sequences of match points (*i.e.*, common subsequences), for $S$ in a search tree and a deterministic finite automaton, respectively, and find the longest common subsequences by traversals of the graphs associated with these structures. If $\mathcal{P}$ is the set of match points for a set $S$ of strings, then the algorithm in [26] runs in $O(k|\Sigma||\mathcal{P}|)$ time and $O(|\mathcal{P}|+kn|\Sigma|)$ space, and the algorithm in [2] runs in $O(kn|\Sigma| \log n + |\mathcal{P}|)$ time and $O((k + |\Sigma|)n + |\mathcal{P}|)$ space.

Observe that in a set $S$ of $k$ $p(o)$-sequences, there can be at most $|\Sigma|o^k$ match points. Therefore, general $p$-LCS is solvable in polynomial time and $p(o)$-LCS is solvable in polynomial time when $k$ and $o$ are small constants. That being said, it is important to note that in the worst case in which all strings in $S$ are length-$n$ strings over a single symbol (*e.g.*, $aaaaaaaa \ldots aaa$), $|\mathcal{P}| = O(n^k)$ and both of these algorithms have time-complexities that are comparable with or even slightly worse than the general dynamic programming algorithm for LCS.

### 2.5.4 Discussion

Table 2.2 summarizes known parameterized results for Longest Common Subsequence, considering parameters of input sequence length, desired LCS length, and number of input sequence, all with respect to different potential restrictions on the alphabet size. The lone remaining open question is the parameterized complexity of

**Table 2.2 The parameterized complexity of Longest Common Subsequence**

| | Alphabet Size $\|\Sigma\|$ | | |
|---|---|---|---|
| Parameter | Unbounded | Parameter | Constant |
| – | NP-hard | $\not\in XP$ | NP-hard |
| $k$ | $W[t]$-hard for $t \geq 1$ | $W[t]$-hard for $t \geq 1$ | $W[1]$-hard |
| $l$ | $W[2]$-hard | FPT | FPT |
| $n$ | ??? | FPT | FPT |

$\langle n \rangle$-LCS, which would allow for many short sequences of unbounded alphabet to be compared and their LCS found.

The previous subsections have demonstrated several exact albeit exponential-time algorithms for LCS. Observe that even though some of these algorithms have very low run times on particular special cases of LCS, all (with the exception of the subsequence enumerate-and-check algorithm) have run times of $O(n^k)$ or greater in the worst case. It is tempting to hope that so-called subexponential algorithms with $O(n^{o(k)})$ running times[4] (*e.g.*, $O(n^{\sqrt{k}})$, $O(n^{\log \log k})$), exist for LCS. However, several recent results make this extremely unlikely, that is,

- When $\Sigma$ is an alphabet of fixed size, LCS is not solvable in $f(k)n^{o(k)}$ time for any function $f$ unless the exponential time hypothesis is false [11].
- LCS is not solvable in $f(l)n^{o(l)}$ time for any function $f$ unless the exponential time hypothesis is false [27, theorem 5].

Note that these results do not forbid exponential-time algorithms whose run times have exponents that are functions of $k$ and/or $l$ and other parameters (*e.g.*, $O(n^{|\Sigma| \log k})$, $O(n^{\log k \sqrt{l}})$), or have bases other than $n$ (*i.e.*, $O(|\Sigma|^k)$, $O(k^l)$). However, these results do suggest that dramatic improvements in general LCS algorithm runtimes will not be forthcoming from the current dynamic programming/edit graph framework, and that future exact algorithm development efforts for LCS (and sequence alignment in general) should explore other options.

## 2.6 COMMON APPROXIMATE SUBSTRING

The most basic formal computational problem associated with sequence consensus is the following:

Common Approximate Substring (CASub(dst))
*Input*: A set $S = \{s_1, s_2, \ldots, s_k\}$ of $k$ strings over an alphabet $|\Sigma|$ and positive integers $l$ and $d$.

[4] In $o()$-notation, $o(g(n))$ is the set of functions $f(n)$ that are asymptotically strictly less than $g(n)$ (*i.e.*, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (cf. Footnote 1)).

*Output*: The string $s'$ of length $l$ such that for every string $s$ in $S$, there is an length $l$ substring $s''$ of $s$ such that $\text{dst}(s', s'') \leq d$.

dst() is a distance-measure function on pairs of strings. Here we will consider the most common of such measures, Hamming distance and edit distance, and their associated problems (CASub(H) and CASub(E)). In an optimization model, with minimizing distance as the objective, the CASub problem also is known as closest substring.

In typical instances of sequence consensus, the sequence-alphabet is of small fixed size (*i.e.*, has $|\Sigma| = 4$ for DNA and RNA sequences and $|\Sigma| = 20$ for protein sequences) the number of sequences $k$ can vary from two to several hundred, the requested substring length can vary from $\leq 25$ to $n$, and the maximum sequence-length (denoted by $n = \max_{s \in S} |s|$) varies from several hundred to several million. This suggests that algorithms that are efficient relative to restrictions on any of these parameters would be of use. However, known intractability results suggest that under many restrictions, polynomial-time efficiency is probably not possible, that is,

- CASub(H) is NP-hard when $|\Sigma| \geq 2$ [21].
- $\langle k, l, d\rangle$-CASub(H) is $W[1]$-hard [18, Theorem 13]; see also [19, Theorem 1].
- $\langle l, d\rangle$-CASub(H) is $W[2]$-hard [18, Theorem 15].
- $\langle k, |\Sigma|\rangle$-CASub(H) is $W[2]$-hard [18, Theorem 20].
- $\langle k, d\rangle$-CASub(H) is $W[1]$-hard when $|\Sigma| = 2$ [34, Theorem 6.1].
- $\langle k\rangle$-CASub(H) is $W[1]$-hard when $|\Sigma| = 2$ [19, Theorem 2].

These hardness results also hold for the arbitrary edit distance cases (E) because Hamming distance is still a potential edit distance. It also may be tempting to consider solution substrings whose lengths are within a small multiplicative factor of the length of the longest substring. Though such approximate substrings can be derived in polynomial time within any constant degree of accuracy [31], the run times are impractical for useful degrees of accuracy; moreover, it is not possible to reduce this run time to make such schemes practical [53]. However, as we will discuss, there yet may be acceptable exact algorithms for special cases.

In the remainder of this section, we will look at algorithms for solving the general common approximate substring problem (Section 2.6.1) and the special case in which all strings in $S$ and the returned string are of the same length (Section 2.6.2).

### 2.6.1 Solving the General Problem

Because of the intractability results, all known exact algorithms run in exponential time. Furthermore, the parameterized hardness results necessitate the inclusion of either the input sequence length $n$ or both the desired substring length $l$ and the alphabet size $|\Sigma|$ in the parameter to have tractable results. Indeed, the number of input sequences $k$ has little effect on the problem's hardness, though if limited, it can be added to other parameters to yield a faster algorithm.

Algorithmic strategies for exactly solving CASub are of three types:

1. Enumerate all $|\Sigma|^l$ strings of length $l$ and check whether each such string is a substring of every string in $S$; return the substring with the smallest maximum distance from strings in $S$ (or, potentially, all strings that appear in each string in $S$).
2. Starting from a short string occurring as a substring of one of the strings in $S$, consider it as a potential CASub result. Gradually modify this string (or another starting string) to accommodate other substrings that are sufficiently close to the developing result until a substring of all strings in $S$ is found.
3. Combine techniques 1 and 2 first to develop gradually a potential common approximate substring and then search its close relatives to adapt it to accommodate other strings. This type of technique often can focus on a limited number of nonidentical columns.

The naive search algorithm applying the first strategy solves $\langle|\Sigma|, l\rangle$-CASub(H) in $O(|\Sigma|^l knl)$ time and $O(kn)$ space [18, Theorem 10(1)]. The modification strategies (strategies 2 and 3) produce a variety of results depending on how the substring is developed and how much of a neighborhood of the developing substring is considered.

For example, the develop center algorithm that implements the second strategy [18, Theorem 10(2)] works by considering each substring of length $l$ of an arbitrary initial string as an instance $C$ of a potential common approximate substring. Because it could have up to $d$ mismatches, all possible $\binom{l}{d}$ selections of $d$ positions in the substring are tried. For each combination, the $d$ positions are replaced by a special blocking character ($\notin \Sigma$), with the remaining unblocked positions occurring exactly in the developing substring. The other strings $s_i$ in $S$ are considered in turn; if $C$ is within distance $d$ of a substring of $s_i$, then $C$ can continue in its current form. If instead there are no substrings of $s_i$ within distance $d$ from $C$, then all substrings of $s_i$ within distance $2d$ are considered, and new alternative substrings $C'$ are created from $C$ by substituting for a minimal number of blocked positions. This process is repeated for each developing substring and each string $s_i$. If the developing process uses all of $S$ for a developed substring, then it reports this substring as a result.

This algorithm solves $\langle n\rangle$-CASub(H) in $O(n^2k\binom{l}{d}(\binom{d}{d/2}n)^d)$ time [18, Theorem 10(2)]. Of particular note in this result is the complete absence of the alphabet size $|\Sigma|$ from the running time; the time is also only linearly dependent on the number of input strings $k$, so it would be the most suitable for applications with a large number of short input strings over an unrestricted (or less restricted) alphabet. It would not be particularly appropriate for DNA or RNA sequences in which the alphabet is very small.

Several different data organization techniques are used to enable algorithms to find similar substrings efficiently and narrow the search of the string neighborhood that they define. These searches often are dependent on the size $N$ of a substring's neighborhood, where $N = \Sigma_{i=1}^{d}\binom{l}{i}(|\Sigma| - 1)^i$. Suffix trees are used and traversed by

Sagot [49] to restrict motif search to $O(lk^2nN)$ time and $O(lk^2n)$ space. A modification of suffix trees to allow errors is used to produce an approximate *neighborhood tree* of common approximate occurrences by Evans and Smith [17] to enumerate all possible CASub results in $O(lknN)$ time and $O(lkn)$ space.

Davilla *et al.* [13] introduce a set of related algorithms that organize their data in lists of the neighboring strings that are kept in lexicographic order and intersected. The neighborhoods are reduced by limiting the search to substrings that are within distance $2d$ of each other because only those substrings can have a common neighbor within distance $d$. This algorithm exploits the computer's word length $w$ as part of a radix sort, and runs in $O(kn^2 + \frac{l}{w}NS)$ time and $O(kn^2)$ space, where $S$ is the sum, over all $\frac{k}{2}$ pairs of consecutive input strings and of the number of substring pairs (one from each string) that are within distance $2d$. [13]. This algorithm also is extended using a branch-and-bound approach to run more efficiently in practice.

Although these results are sufficient to resolve the parameterized complexity of all parameter combinations and provide some different tradeoffs between the parameters, incorporating additional parameters greatly can improve the best known running times for algorithms that solve the problem, and they can be exploited by different data organization and search space-narrowing techniques. Marx [34] developed two different techniques for motif search using $\langle|\Sigma|, n, d\rangle$ as parameters, with the second technique also including $k$ as part of the parameter for additional efficiency. Without limiting $k$, a common approximate substring can be found by considering the substrings of strings in $S$ that generate it by their identical positions; all length $l$ substrings occurring in $S$ are considered, and Marx proved that considering only substring subsets of size $\leq \log_2 d + 2$ are sufficient to generate any possible common approximate substring (if one exists). The remaining positions in the solution can be found through exhaustive search, yielding an algorithm that runs in $O(|\Sigma|^{d(\log d+2)}n^{\log d+O(1)})$ time [34, Theorem 2.3].

Ma and Sun reduce this running time by providing a $O(kl + kd2^{4d}|\Sigma|^d n^{\lceil \log d\rceil+1})$ time [32, Theorem 2] algorithm, which operates by repeatedly modifying an arbitrarily chosen substring, defining some positions as error-free and searching through other possible characters for the remaining positions.

Faster techniques are possible if $k$ is also limited and included in the parameter. For this situation, Marx [34] builds a hypergraph with the $l$ possible positions in a substring as its vertices; a hyperedge is added for each substring $s_i'$, linking those positions in that substring that are different from a selected base substring $s_1'$. Each substring occurring in $s_1$ is used in turn as the base substring for constructing such a hypergraph. A common approximate substring can be found by considering all occurrences of half-covering subhypergraphs, which are constant in number and each have a $O(\log\log k)$ fractional cover number. Their enumeration then solves the problem in $O((|\Sigma|d)^{O(kd)}n^{O(\log\log k)})$ time [34, Theorem 4.5].

### 2.6.2 Special Case: Common Approximate String

For many applications of sequence consensus in computational biology, the entirety of each input sequence needs to be covered by a full-length consensus sequence. This restriction of $l = n$ produces a well-investigated special case of CASub, Common Approximate String (CAStr), better known by its optimization name Closest String. As with CASub, CAStr can incorporate different distance measures including Hamming distance (H) and edit distance (E). This restriction on the problem makes all parameterized variants that include either $d$ or $k$ as a parameter tractable for CAStr under Hamming distance despite the intractability of the corresponding CASub variants. Some intractability results, however, still hold, especially if an arbitrary edit distance is to be used, that is

- CAStr(H) is NP-hard when $|\Sigma| \geq 2$ [21].
- CAStr(E) is NP-hard when $|\Sigma| \geq 2$ [41, Theorem 3]; result also holds under arbitrary weighted edit distance [41, theorem 6]
- $\langle k \rangle$-CAStr(E) is $W[1]$-hard when $|\Sigma| \geq 2$ [41, Theorem 3]; result also holds under arbitrary weighted edit distance [41, theorem 6]

Though $\langle d \rangle$-CASub is $W[1]$-hard, the corresponding variant of CAStr is in FPT for Hamming distance. Gramm *et al.* use a linear search tree to solve this problem in $O(kn + kd^{d+1})$ time [24, Theorem 1]. In this strategy, a consensus string is searched for by repeatedly picking a string that is not sufficiently close to the current prospective solution and then modifying the solution to bring the string into the neighborhood. They also show that $\langle k \rangle$-CAStr(H) is FPT by describing how to construct an integer linear program with no more than $B(k) \times k$ variables, where $B(k) < k!$ is the $k$th Bell number. Although the running time grows very quickly with respect to $k$, it is however linear with respect to the input size [24, Theorem 4]. Restricting $l$, potentially useful for arbitrarily sized alphabets, has the effect of restricting $d$, so CAStr(H) is thus fixed-parameter tractable for all parameters except $|\Sigma|$ alone.

Adding $|\Sigma|$ to the parameter enables a faster algorithm for those cases in which the alphabet is small. Ma and Sun [32] use a similar technique for CAStr as they do for CASub; indeed, the CAStr algorithm forms the basis for their CASub algorithm that also needs to consider different substrings of the input strings. Eliminating this need greatly simplifies the running time needed, making the result only linearly dependent on the string length $n$, thus yielding an algorithm that runs in $O(nk + kd(16|\Sigma|)^d)$ time [32, Corollary 1].

### 2.6.3 Discussion

Table 2.3 summarizes the known parameterized results for Common Approximate Substring. Most work so far has focused on the Hamming distance versions of these problems; these results should be used as a basis for further exploration of more general edit distance, likely considering different restrictions on distance such as metrics. The work of [32, 34] also could be extended to find faster algorithms for the variants known to be in FPT and for even faster algorithms when additional problem aspects can be restricted and included in the parameter. Note, however, that there are known limitations on such algorithm development, as there are no $f_1(k, d)n^{o(\log d)}$

**Table 2.3 The parameterized complexity of Common Approximate Substring**

| | Alphabet Size $\|\Sigma\|$ | | |
|---|---|---|---|
| Parameter | Unbounded | Parameter | Constant |
| – | NP-hard | $\not\in$ XP | NP-hard |
| $k$ | $W[2]$-hard | $W[2]$-hard | $W[1]$-hard |
| $d$ | $W[2]$-hard | $W[1]$-hard | $W[1]$-hard |
| $l$ | $W[2]$-hard | FPT | FPT |
| $n$ | FPT | FPT | FPT |

or $f_2(k, d)n^{o(\log \log k)}$ time algorithms for CASub(H) unless the exponential time hypothesis fails [34, Corollary 6.8].

As for CAStr, CAStr(H) inherits FPT results from CASub(H) and has additional FPT results for parameters that make CASub(H) intractable, completing CAStr(H)'s parameterized complexity map. Many of these algorithms, however, have high exponential functions of the parameter, so further development is needed to produce useful algorithms. Examining CAStr relative to more general edit distances also should produce interesting results.

## 2.7 CONCLUSION

The results outlined in the preceding sections show that fixed-parameter algorithms and other special cases can solve problems that generally are considered intractable, providing solutions that are consistent with the problems in computational biology to which the theoretical problems are applied.

Parameterized results are usually only a starting point for research. Once a variant has been shown to be in FPT for its parameter set, the algorithm usually can be made more efficient through incorporating additional fixed-parameter techniques. The development of better algorithms for Common Approximate Substring as described in Section 1.6.1 is a good example of this type of work; these algorithms also show that, when multiple characteristics of the problem are included in the parameter, different approaches and their respective parameter tradeoffs may be more or less appropriate depending on the parameter restrictions and values characteristic of the specific applications. Regardless of such parameterized efforts, it is critical that work also be done on restricted special cases characterized by structural restrictions because some problem characteristics cannot be captured well or at all by parameterized restrictions.

The work presented in this chapter demonstrates how application-focused problem analysis has been successful at finding tractable and useful special cases for basic sequence problems. Given this success, this work should be continued for the problems discussed here and, perhaps more importantly, extended to other problems in computational biology.

## REFERENCES

1. R. Arratia, D. Martin, G. Reinert, and M.S. Waterman. Poisson process approximation for sequence repeats, and sequencing by hybridization. *J Comput Biol*, 3:425–464, 1996.
2. R. Baeza-Yates. Searching subsequences. *Theor Comput Sci*, 78:363–376, 1991.
3. M. Barsky, U. Stege, A. Thomo, and C. Upton. Shortest path approaches for the longest common subsequence of a set of strings. *Proceedings of the 7th International Symposium on Bioinformatics and Bioengineering (BIBE'07)*, IEEE Computer Society, New York, 2007, pp. 327–333.
4. S. Bereg, M. Kubica, T. Walent, and B. Zhu. RNA multiple structural alignment with longest common subsequences. *J Combin Optim*, 13(2):178–188, 2007.
5. L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. *Proceedings of the 7th International Symposium on String Processing Information Retrieval (SPIRE'00)*, IEEE Computer Society, New York, 2000, pp. 39–48.
6. J. Błazewicz, P. Formanowicz, M. Kasprzak, W.T. Markiwicz, and J. Weglarz. DNA sequencing with positive and negative errors. *J Comput Biol*, 6(1):113–123, 1999.
7. J. Błazewicz and M. Kasprzak. Complexity of DNA sequencing by hybridization. *Theor Comput Sci*, 290:1459–1473, 2005.
8. A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *J ACM*, 41(4):630–647, 1994.
9. H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, and H.T. Wareham. Parameterized complexity analysis in computational biology. *Comput Applic Biosci*, 11(1):49–57, 1885.
10. H.L. Bodlaender, R.G. Downey, M.R. Fellows, and H.T. Wareham. The parameterized complexity of sequence alignment. *Theor Comput Sci*, 147:31–54, 1994.
11. J. Chen, X. Huang, I. Kanj, and G. Xia. W-hardness linear FPT reductions: structural properties and further applications. *Proceedings of the 11th International Computing and Combinatorics Conference (COCOON 2005)*, Lecture Notes in Computer Science no. 3595, Springer, New York, 2005, pp. 975–984.
12. K.M. Chao, W.R. Pearson, and W. Miller. Aligning two sequences within a specific diagonal band. *Comput Applic Biosci*, 8:481–487, 1992.
13. J. Davilla, S. Balla, and S. Rajesakaran. Fast and Practical Algorithms for Planted $(l, d)$ Motif Search. *IEEE/ACM Trans Comput Biol Bioinform*, 4(4):544–552, 2007.
14. W.H.E. Day and F.R. McMorris. The computation of consensus patterns in DNA sequences. *Math Comput Model*, 17:49–52, 1993.
15. R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York, 1999.
16. M. Dyer, A. Frieze, and S. Suen. The probability of unique solutions of sequencing by hybridization. *J Comput Biol*, 1:105–110, 1994.
17. P.A. Evans and A.D. Smith. Toward optimal motif enumeration. *Proceedings of the 8th International Workshop on Algorithms and Data Structures (WADS 2003)* Lecture Notes in Computer Science no. 2748, Springer, New York, 2003, pp. 47–58.
18. P.A. Evans, A.D. Smith, and H.T. Wareham. On the complexity of finding common approximate substrings. *Theor Comput Sci*, 306:407–430, 2003.
19. M.R. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of motif search problems. *Combinatorica*, 26(2):141–167, 2006.

20. M.R. Fellows, M.T. Hallett, and U. Stege. Analogs & duals of the MAST problem for sequences and trees. *J Algorithm*, 49:192–216, 2003.
21. M. Frances and A, Litman. On covering problems of codes. *Theor Comput Syst*, 30(2): 113–119, 1997.
22. J. Gallant, D. Maier, and J.A. Storer. On finding minimal length superstrings. *J Comput Syst Sci*, 20:59–58, 1980.
23. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1999.
24. J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37:25–42, 2003.
25. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997.
26. W.J. Hsu and M.W. Du. Computing a longest common subsequence for a set of strings. *BIT*, 24:45–59, 1984.
27. X. Huang. Lower Bounds and Parameterized Approach for longest Common Subsequence. *Proceedings of the 12th International Conference on Computing and Combinatorics (COCOON 2006)*, Lecture Notes in Computer Science no. 4112, Springer, New York, 2003, pp. 136–145.
28. R.W. Irving and C.C. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM'93)*, Lecture Notes in Computer Science no. 644, Springer, New York, 2003, pp. 214–229.
29. T. Jiang and M. Li. On the Approximation of Shortest Common Supersequences and Longest Common Subsequences. *SIAM J Comput*, 24(5):1122–1139, 1995.
30. T. Jiang, M. Li, and Z.-z. Du. A note on shortest superstrings with flipping. *Inf Process Lett*, 44:195–199, 1992.
31. M. Li, B. Ma, and L. Wang. On the Closest String and Substring Problems. *J ACM*, 49(2): 151–171, 2002.
32. B. Ma and X. Sun. More Efficient Algorithms for Closest String and Substring Problems. *Proceedings of the 12th Annual International Conference on Research in Computational Biology (RECOMB 2008)*, Lecture Notes in Computer Science no. 4955, Springer, New York, 2008, pp. 296–409.
33. D. Maier. The complexity of some problems on subsequences and supersequences. *J ACM*, 25:322–336, 1978.
34. D. Marx. The Closest Substring problem with small distances. *Proceedings of the 46th Annual Symposium on Foundations of Computer Science (FOCS 2005)*, IEEE Computer Society, 2005, pp. 1–10.
35. P. Medvedev, K. Georgiou, G. Myers, and M. Brudno. Computability of models for sequence assembly. *Proceedings of the 7th International Workshop on Algorithms in Bioinformatics (WABI 2007)*, Lecture Notes in Computer Science no. 4645, Springer, New York, 2007, pp. 289–301.
36. M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theor Comput Sci*, 125:205–228, 1994.
37. M. Middendorf. Shortest common superstring and scheduling with coordinated starting times. *Theor Comput Sci*, 191:205–214, 1998.
38. W. Miller and E.W. Myers. A file comparison program. *Software—Pract Exp*, 15(1): 1035–1040, 1985.

39. B.M.E. Moret, J. Tang, and T. Warnow. Reconstructing phylogenies from gene-content and gene-order data. In O. Gascuel editor, *Mathematics of Phylogeny and Evolution.* Oxford University Press, New York, 2004.

40. E.W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(Sup2), ii79–ii85, 2005.

41. F. Nicolas and E. Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *J Discrete Algorithm*, 3:390–415, 2005.

42. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, New York, 2006.

43. K. Ning, H. Kee Ng, and H. Wai Leong. Finding patterns in biological sequences by longest common subsequences and shortest common supersequences. *Proceedings of the 6th International Symposium on Bioinformatics and Bioengineering (BIBE'06)*, IEEE Computer Society, New York, 2006, pp. 53–60.

44. S. Ott. Bounds for approximating shortest superstrings over an alphabet of size 2. *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99)*, Lecture Notes in Computer Science no. 1665, Springer, New York, 1999, pp. 55–64.

45. P.A. Pevzner. Multiple alignment, communication cost, and graph matching. *SIAM J Appl Math*, 52:1763–1779, 1992.

46. P.A. Pevzner, H. Tang, and M.S, Waterman. An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci*, 98:9748–9753, 2001.

47. K. Pietrzak. On the parameterized complexity of the fixed alphabet *shortest common supersequence* and *longest common subsequence* problems. *J Comput Syst Sci*, 67:757–771, 2003.

48. M. Pop. Genome assembly reborn: recent computational challenges. *Briefings Bioinf*, 10(4):354–366, 2009.

49. M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. *Proceedings of the Third Latin American Symposium on Theoretical Informatics* Lecture Notes in Computer Science no. 1390, Springer, New York, 1998, pp. 374–390.

50. Z. Sweedyk. A $2\frac{1}{2}$-approximation algorithm for shortest superstring. *SIAM J Comput*, 29(3):954–986, 1999.

51. V.G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25:565–580, 1990; translated from *Kibernetica*, 25:1–13, 1989.

52. V. Vassileska. Explicit inapproximability bounds for the shortest common superstring problem. *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005)*, Lecture Notes in Computer Science no. 3618, Springer, New York, 2005, pp. 793–800.

53. J. Wang, J. Chen, and M. Huang. An improved lower bound on approximation algorithms for the Closest Substring problem." *Inf Process Lett*, 107:24–28, 2008.

54. M.S. Waterman. *Introduction to Computational Biology*. Chapman and Hall, New York, 1995.

55. S. Wu, U. Manber, G. Myers, and W. Miller. An $O(NP)$ sequence comparison algorithm. *Inf Process Lett*, 35:317–323, 1990.

# 3

# FINITE AUTOMATA IN PATTERN MATCHING

Jan Holub

## 3.1 INTRODUCTION

Stringology is a part of computer science dealing with processing strings and sequences. It finds many important applications in various fields used by information society. Biological sequences processing is one of the most important fields. However, the finite automata theory is a well-developed formal system used for a long time in the area of compiler construction.

The chapter aims to show various approaches of the finite automata use in stringology. The approaches are demonstrated on practical examples. Of course, it is impossible to describe all approaches, as it would be out of scope of the chapter. However, we would like to present basic approaches that the reader can modify and combine to a given task.

The following four kinds of finite automata use were identified in the area of stringology:

1. A direct use of deterministic finite automata (DFA)
2. A simulation of nondeterministic finite automata (NFA)
3. A use of finite automata as a model for computation
4. A composition of various automata approaches for particular subproblems

The direct use of DFA is used in case of pattern matching automata when a DFA is built over the given pattern and then a text is given as an input to the DFA (the pattern

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

is preprocessed). One also can construct a DFA over a given text and a pattern is given as an input of DFA, which is the case of factor automata (providing a complete index of the text; the text is preprocessed). Which of these approaches is used depends on the real task and what data we have in advance.

If the pattern matching DFA is too large, then one can use the NFA simulation. Three simulation methods are presented: basic simulation method (BSM), bit parallelism (BP), and dynamic programming (DP). BSM is a general simulation method that works for any NFA. BP and DP improve running times of BSM in a special case when NFA has a regular structure (like in pattern matching).

When an NFA is constructed for a given task and then determinized, the structure of the resulting DFA can bring answers to some questions. Particularly, so-called d-subsets are studied for identifying both exact and approximate repetitions.

More and more complicated problems appear in stringology. If we can decompose such a complicated problem into simple subproblems and solve these subproblems by automata approaches, then we can build the resulting solution as a composition of solutions of the subproblems.

### 3.1.1 Preliminaries

Let $\Sigma$ be a nonempty input alphabet, $\Sigma^*$ be the set of all strings over $\Sigma$, $\varepsilon$ be the *empty string*, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. If $a \in \Sigma$, then $\overline{a} = \Sigma \setminus \{a\}$ denotes a *complement* of $a$ over $\Sigma$. If $w = xyz$, $x, y, z \in \Sigma^*$, then $x, y, z$ are *factors* (substrings) of $w$; moreover, $x$ is a *prefix* of $w$ and $z$ is a *suffix* of $w$.

NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where $Q$ is a set of states, $\Sigma$ is a set of input symbols, $\delta$ is a mapping $Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^{|Q|}$, $q_0$ is an initial state, and $F \subseteq Q$ is a set of final (accepting) states. DFA is NFA, where $\delta$ is a mapping $Q \times \Sigma \mapsto Q$. We can extend $\delta$ to $\hat{\delta}$ mapping $Q \times \Sigma^* \mapsto 2^{|Q|}$ for NFA or $Q \times \Sigma^+ \mapsto Q$ for DFA, respectively. DFA (respectively, NFA) accepts a string $w \in \Sigma^*$ if and only if $\hat{\delta}(q_0, w) \in F$ (respectively, $\hat{\delta}(q_0, w) \cap F \neq \emptyset$). The set of all strings accepted by automaton $M$ is the language of automaton denoted by $L(M)$. For more details see [3].

If $P \subseteq Q$, then for NFA, we define $\varepsilon\text{CLOSURE}(P) = \{q' \mid q' \in \hat{\delta}(q, \varepsilon), q \in P\} \cup \{P\}$. In other words, $\varepsilon$CLOSURE$(P)$ contains all states accessible from states in $P$ using only $\varepsilon$-transitions.

An *active state* of NFA, when the last symbol of a prefix $w$ of an input string is processed, denotes each state $q$, $q \in \hat{\delta}(q_0, w)$. At the beginning, only $q_0$ is an active state.

A *depth of state* $q$ in NFA is the minimum number of moves that are needed to get from an initial state $q_0$ to this state $q$ without using $\varepsilon$-transitions. A *level of state* $q$ in NFA is the minimum among the numbers of differences (errors) associated with all final states reachable from $q$. In the figures of this chapter, the states of the same depth are in the same column, and the states of the same level are in the same row.

An algorithm $\mathcal{A}$ simulates a run of an NFA; if $\forall w$, $w \in \Sigma^*$, then it holds that $\mathcal{A}$ with given $w$ at the input reports all information associated with each final state $q_f$, $q_f \in F$, after processing $w$ if and only if $q_f \in \hat{\delta}(q_0, w)$.

**Table 3.1 Basic classification of pattern matching algorithms using finite automata**

| | | Text | |
|---|---|---|---|
| | | Not Preprocessed | Preprocessed |
| Pattern | Not Preprocessed | Elementary algorithm | Suffix, factor, and oracle automata |
| | Preprocessed | Pattern matching automata | Intersection of automata |

For measuring the similarity of two strings $v, w \in \Sigma^*$, we use edit distance $D(v, w)$ defined as the minimum number of edit operations needed to convert $v$ to $w$. We distinguish three edit distances:

1. *Hamming distance* [18] $D_H$, which allows the edit operation *replace*
2. *Levenshtein distance* [33] $D_L$, which allows *replace*, *insert*, and *delete*
3. *Damerau distance* [15] (also called *generalized Levenshtein distance*) $D_D$, which allows *replace*, *insert*, *delete*, and *transpose*. Each symbol of $v$ can participate at most in one edit operation *transpose*.

In exact string matching, one can look at the automata solutions used according to a preprocessing text and/or pattern as shown in Table 3.1. When neither pattern nor text is preprocessed, an elementary (naive) search is performed employing a lot of comparisons. If we preprocess the pattern, then we get classical pattern matching automaton running on the input text. On the other hand, if we preprocess the text, then we get suffix, factor, or oracle automata. If we consider preprocessing both the pattern and the text, then we get two automata (one for pattern, one for text) and we try to find the so-called intersection automaton (approach number 4). The previously mentioned approaches can be applied also to the approximate string matching; however, the complexity then rises.

## 3.2 DIRECT USE OF DFA IN STRINGOLOGY

### 3.2.1 Forward Automata

Traditional finite automata are accepting automata. They read whole input text $w$, and then text $w$ is accepted (verified) if the finite automaton reaches a final state. In other words, $w$ is in the set of strings (called language) accepted by the finite automaton. In addition, *pattern matching automaton* (see Figure 3.1) has been designed in stringology. It traverses the input text $w$ and reports each location of a given pattern $p$ (*i.e.*, the finite automaton accepts language $L = \{up \mid u \in \Sigma^*\}$, checking any prefix of $w$). So in each step, the pattern matching automaton checks whether a final state is reached. The verification (accepting) automaton performs the check only once at the end of the text traversal. From the biology point of view,

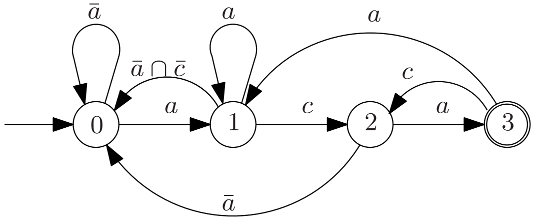


**Figure 3.1** DFA for the exact string matching ($p$ = aca).

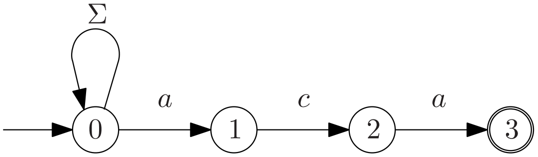


**Figure 3.2** NFA for the exact string matching ($p$ = aca).

the text may be a DNA sequence (alphabet $\Sigma$ is then $\{a, c, g, t\}$), and the pattern may be a gene we would like to find in the DNA sequence. The text and pattern also may be build over the symbols representing amino acids. The alphabet is then $\Sigma = \{g, a, v, l, i, p, f, y, c, m, h, k, r, w, s, t, d, e, n, q, b, z\}$.

One easily can construct an NFA first (see Figure 3.2) and transform it into the equivalent DFA using the standard subset construction [26, 30]. The determinization runs in time $\mathcal{O}(|Q_{\text{DFA}}||Q_{\text{NFA}}||\Sigma|)$, where $|Q_{\text{NFA}}|$ and $|Q_{\text{DFA}}|$ are numbers of states of the NFA and the resulting DFA, respectively. The direct DFA construction algorithm also exists [12] (see Algorithm 3.1), running in time $\mathcal{O}(m|\Sigma|)$, where $m$ is the length of the pattern.

**Algorithm 3.1 (Construction of DFA for the exact string matching)**
**Input:** Pattern $p = p_1p_2 \ldots p_m$.
**Output:** DFA $M$ accepting language $L(M) = \{wp \mid w \in \Sigma^*\}$.
**Method:** DFA $M = (\{q_0, q_1, \ldots, q_m\}, \Sigma, \delta, q_0, \{q_m\})$, where the mapping $\delta$ is constructed as follows:

```
for each $a \in \Sigma$ do
    $\delta(q_0, a) \leftarrow \{q_0\}$                    /* self-loop of the initial state */
endfor
for $i \leftarrow 1, 2, \ldots, m$ do
    $r \leftarrow \delta(q_{i-1}, p_i)$
    $\delta(q_{i-1}, p_i) \leftarrow q_i$                  /* forward transition */
    for each $a \in \Sigma$ do
        $\delta(q_i, a) \leftarrow \delta(r, a)$
    endfor
endfor
```

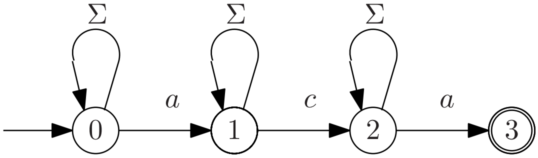


**Figure 3.3** NFA for the exact sequence matching ($p$ = aca).

Well, the pattern does not need to be a solid string. One can consider pattern $p = p_1 p_2 \ldots p_m$ as a sequence. Thus, one searches for all occurrences of $p$ in input text $w$ where any number of symbols are inserted between any adjacent pattern symbols $p_i$ and $p_{i+1}$. The NFA for exact sequence matching can be constructed easily from the NFA for exact string matching by inserting a self loop labeled by $\Sigma$ (matching any symbol of alphabet) to any nonfinal state as shown in Figure 3.3. The self-loop skips those inserted symbols.

In biology, when we search for genes, we actually search for a sequence of exons (coding sequences) while skipping introns (noncoding sequences) by self-loops labeled by $\Sigma$ labels. The NFA for a gene searching is shown in Figure 3.4. The gene consists of three exons aca, tag, and gaa.

Another extension is approximate string matching in which we allow up to $k$ errors where $k$ is given. The errors can be introduced by edit operations *replace*, *delete*, *insert*, and *transpose*. Let us consider just *replace*, *insert*, and *delete*, which are the base for the Levenshtein distance. The corresponding approximate string matching NFA for pattern $p$ is constructed as $k + 1$ copies ($M_0, M_1, \ldots, M_k$) of exact string matching automaton for pattern $p$: one for "no error" (level 0), one for "one error" (level 1), ..., one for "$k$ errors" (level $k$). These $k + 1$ automata are connected by the transitions representing edit operations.

Each transition for *replace* is labeled by symbol $\overline{p}_{j+1}$ (mismatching symbol $p_{j+1}$ in pattern $p$) and leads from state $q_j$ of automaton $M_i$ to state $q_{j+1}$ of automaton $M_{i+1}$, $0 \le i < k$, $0 \le j < m$; the depth $j$ in the automaton (corresponding to a pointer in the pattern) is increased as well as the minimum number of errors. Each transition for *insert* is labeled by symbol $\overline{p}_{j+1}$ and leads from state $q_j$ of automaton $M_i$ to state $q_j$ of automaton $M_{i+1}$, $0 \le i < k$, $0 < j < m$; the minimum number of errors is increased, but the depth in automaton remains the same. Each transition for *delete* is labeled by $\varepsilon$ and leads from state $q_j$ of automaton $M_i$ to state $q_{j+1}$ of automaton $M_{i+1}$, $0 \le i < k$, $0 \le j < m$; the depth in automaton is increased as well as the minimum number of errors, but no symbol is read from the input.

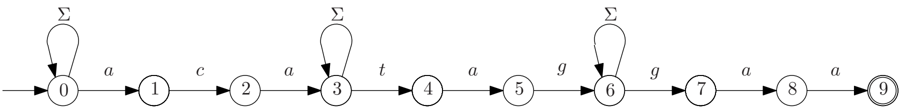


**Figure 3.4** NFA for the exact gene matching (gene has three exons aca, tag, and gaa).

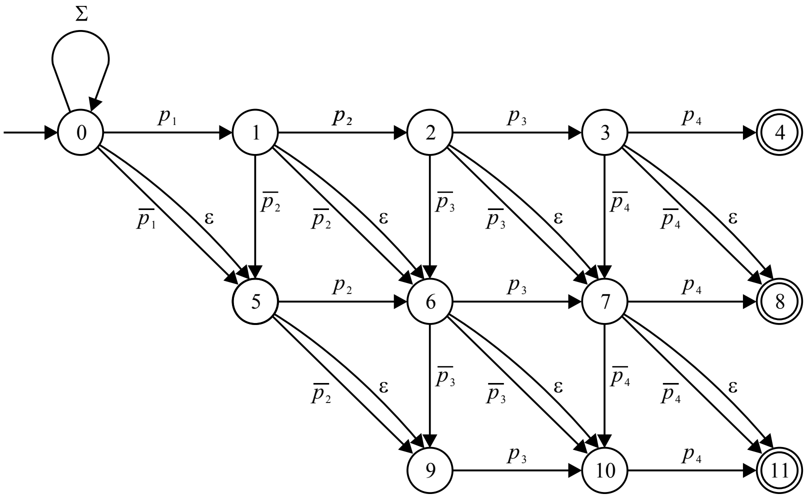


**Figure 3.5** NFA for the approximate string matching using Levenshtein distance ($m = 4$, $k = 2$).

The initial state of such automaton is the initial state of automaton $M_0$. Inaccessible states of the resulting automaton are removed. The resulting NFA is shown in Figure 3.5.

After the determinization of this NFA, the resulting DFA may have up to $(k + 1)!(k + 2)^{m-2}$ states [34]. This gives us the memory complexity of DFA and its preprocessing time. The following DFA run then has the time complexity linear with the size of the input text. However, if the DFA is too large, then one has to use a NFA simulation described in Section 3.3.

The previous NFA has the same weights of all edit operations. If the weights are different integers, then we just need to reconnect the corresponding transitions to states of corresponding levels.

### 3.2.2 Degenerate Strings

In bioinformatics, some positions of DNA are not determined precisely, or we do not care which symbol of a subset of alphabet is present. The position simply matches more than one symbol of alphabet. In such cases, we talk about a *degenerate symbol*. Special symbols for those degenerate symbols (e.g., $H$ matches $a$, $c$, or $t$) are introduced in [39]. A string containing one or more degenerate symbols is called a *degenerate string*. It also is called an *indeterminate* or a *generalized string*. For example, isoleucine is the amino acid encoded by triplet $atH$.

NFA can handle degenerate symbols very easily. Instead of having a single matching transition, we introduce one transition for each matching symbol as shown in Figure 3.6. We usually collapse the parallel transitions into one transition labeled by several symbols. For exact indeterminite string matching, a sublinear algorithm exists [24].

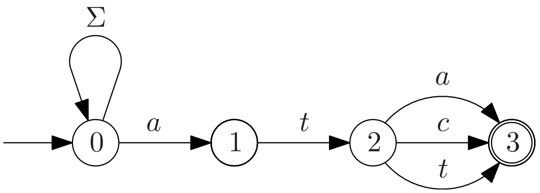


**Figure 3.6** NFA searching all occurrences of isoleucine ($p = atH$).

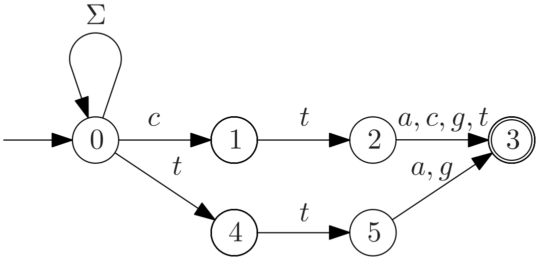


**Figure 3.7** NFA searching all occurrences of leucine.

A single degenerate string cannot be used for all amino acids. For example, leucine is encoded as "$ctN$ and $ttR$" (*i.e.*, "$ct\{a, c, g, t\}$ and $tt\{a, g\}$"). If we encode it as $YtN = \{t, c\}t\{a, c, g, t\}$, then it also covers phenylalanine encoded as $ttY = tt\{c, t\}$. Thus, when searching for occurrences, we have to specify the pattern as "$ctN$ or $ttR$." NFA solves this problem very easily as well. Using Thompson's construction [43], we get the NFA shown in Figure 3.7.

The same degenerate string approach can be used when searching for an amino acid pattern in an amino acid text. For example, as isoleucine and leucine have the same chemical properties, we may not care which actually is found on a given position of the pattern.

So far, all algorithms presented were constructing NFA first, then converting it into DFA, and finally running the DFA over the text in time $\mathcal{O}(n)$ where $n$ is the length of the input text. The preprocessing time (NFA construction and determinization) and space is linear with the pattern size. For approximate pattern matching, the preprocessing time and space is worse, as already stated.

### 3.2.3 Indexing Automata

Another approach used in stringology is to preprocess the text and not the pattern; we build an automaton for the input text and give the pattern as an input to the finite automaton. This is the case of factor and suffix automata[1] [9, 10, 14]. We build a suffix or factor automaton for a given text $w$, and then in $m$ steps, we figure out

[1] This kind of automaton also is called directed acyclic word graph (DAWG). However, then it is not clear whether DAWG is the suffix or the factor automaton.

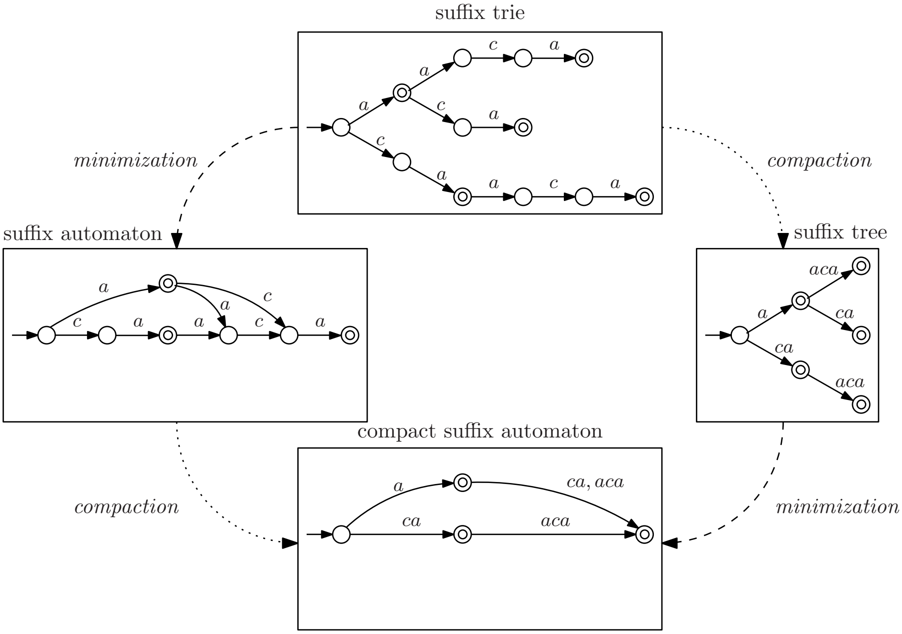


**Figure 3.8** Relation among suffix trie, suffix tree, suffix automaton, and compact suffix automaton ($p$ = caaca).

whether a given pattern $p$ of length $m$ is a factor (substring) of text $w$. Therefore, the automaton also is called indexing automaton (or complete index) of the text. This process is a very efficient full-text searching in a collection of documents in which the documents are given in advance while the pattern is provided at the moment when we want to know in which documents it is located.

The suffix automaton is the minimal DFA accepting all suffixes of a given string $w$. The factor automaton is the minimal DFA accepting all factors of $w$. However, the suffix automaton is a little bit simpler to construct, and it also can identify all factors of $w$. Therefore, the suffix automaton is used in practice. The suffix automaton has at most $2n - 1$ states [12].

Suffix trie and suffix tree are other indexing automata that are less space efficient. However, the most space efficient is compact suffix automaton. Their mutual relation is shown in Figure 3.8.

The most efficiently implemented data structure for indexing an input text is the suffix tree by Kurtz [32], suffix automaton by Balík [8], and compact suffix automaton by Holub and Crochemore [21]. The implementations of suffix automaton and compact suffix automaton are very close in space requirements—ranges about 1–5 bytes[2] per input symbol. However, the implementation of suffix automaton [8] is

[2] The value depends on the type of the input text.

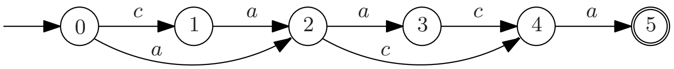


**Figure 3.9** Factor oracle ($p$ = caaca).

focused on the smallest space used (using some compression techniques), the implementation of compact suffix automaton [21] is focused on the speed of traversing.

The next section describes even more compacted automaton called factor oracle [4] (Figure 3.9).

### 3.2.4 Filtering Automata

One can use pattern matching DFA as a filter. An example of such technique is factor oracle [4]—an automaton build over an input text $w$ of size $n$. The factor oracle is smaller than the factor automaton. It has only $n + 1$ states. However, in addition to all factors of input text $w$, it also accepts some other strings as a penalty for only $n + 1$ states. Therefore, the answer "no" means the pattern $p$ is not a factor of $w$, and the answer "yes" means "maybe," so it should be verified.

### 3.2.5 Backward Automata

Backward matching algorithms align the pattern at the first possible position in the text and start to compare the pattern and the text backward. If a mismatch is found, then the pattern is shifted further and a new comparison begins. Thus, these methods can achieve a sublinear time.

The first algorithm using this approach was the Boyer-Moore algorithm [11]. There are many variants of the algorithm like the Boyer-Moore-Horspool [27] or the Boyer-Moore-Sunday [42], which perform a pure comparison, and the following shift is done according to the last aligned symbol or the symbol after, respectively.

The suffix automaton can be applied to a pattern in backward matching as well. In backward DAWG matching algorithm (BDM) [13], the reversed suffix automaton is constructed for reversed pattern. Running the automaton backward identifies possible prefixes that influence the next shift of the pattern. Nondeterministic automaton for BDM called BNDM (backward nondeterministic DAWG matching; see Figure 3.10) [38] and backward oracle matching (BOM) algorithm [4] also work backward and they are very efficient. They use the bit parallel NFA simulation technique discussed in Section 3.3.2.

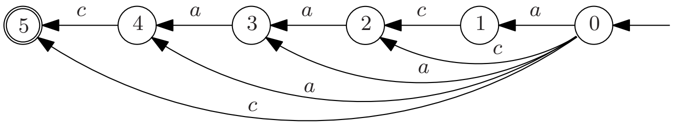


**Figure 3.10** BNDM automaton ($p$ = caaca).

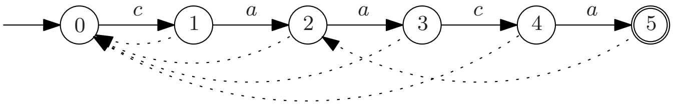


**Figure 3.11** KMP automaton ($p$ = caaca).

### 3.2.6 Automata with Fail Function

For all aforementioned finite automata, the DFAs are used. If we construct NFA, then we have to transform it to the equivalent DFA. A special kind of DFA also is used in stringology. Knuth-Morris-Pratt (KMP) type automaton [29][3] is a special DFA extended by fail function (see Figure 3.11). KMP does not have all outgoing transitions defined (it is called not completely defined automaton). If the automaton should use an undefined transition, then the fail function is used to switch the automaton into another state. The fail function is used until we reach a state with the required transition defined. In the worst case, the initial state is reached in which the transitions are defined for all symbols of the input alphabet. KMP automaton is very memory efficient, it needs memory linear to the length of the pattern. KMP automaton is used for exact string matching for one pattern. Aho-Corasick (AC) type automaton [2] is an extension of KMP for a finite set of patterns.

## 3.3 NFA SIMULATION

DFA cannot always be used. Theoretically, we can face up to an exponential increase of number of states when determinizing NFA. If the resulting DFA cannot fit into memory, then we have to search for another approach. Thus, the NFA simulation is used. Instead of determinizing, we traverse the NFA in width-first order with a goal to reach a final state. The basic simulation method [19, 20] was designed for that purpose. It was implemented using bit vectors.

For NFA with a regular structure, we also can use other simulation methods called bit parallelism and dynamic programming. They improve time and space complexities; however, they cannot be used for general NFA.

NFA simulation runs slower than DFA, but it requires less memory. NFA simulation also is used when determinization would take too much time with respect to the length of input text. A resolution system [25] was developed that for a given stringology task recommends the most suitable method (DFA or one of simulation methods).

To speed up the BSM, the deterministic state cache was implemented [22]. It combines advantages of both DFA run and NFA simulation as shown in Section 3.3.4.

Once we know how to simulate NFA efficiently, we can use this approach for other stringology tasks like BNDM [38] and BOM [4] for exact pattern matching.

[3] KMP automaton [29] is an optimized version of the original Morris-Pratt (MP) automaton [36].

### 3.3.1 Basic Simulation Method

The basic simulation method maintains a set $S$ of active states during the whole simulation process. At the beginning, only the state $q_0$ is active, and then we evaluate $\varepsilon$-transitions leading from $q_0$: $S_0 = \varepsilon\text{CLOSURE}(\{q_0\})$. In the $i$-th step of the simulation with text $t = t_1t_2\dots t_n$ on input (*i.e.*, $t_i$ is processed), we compute a new set $S_i$ of active states from the previous set $S_{i-1}$ as follows: $S_i = \bigcup_{q\in S_{i-1}} \varepsilon\text{CLOSURE}(\delta(q, t_i))$. In each step, we also check whether $S_i = \emptyset$ then the simulation finishes (*i.e.*, NFA does not accept $t$) and whether $S_i \cap F \neq \emptyset$, then we report that a final state is reached (*i.e.*, NFA accepts string $t_1t_2\dots t_i$). If each final state has an associated information, then we report it as well.

This simulation is implemented by using bit vectors as described in [19]. This implementation runs in time $\mathcal{O}(n|Q|\lceil\frac{|Q|}{w}\rceil)$ and space $\mathcal{O}(|\Sigma||Q|\lceil\frac{|Q|}{w}\rceil)$, where $w$ is a length of used computer word in bits, $|Q|$ is a number of states of NFA, and $n$ is a length of the input string.

### 3.3.2 Bit Parallelism

The bit parallelism [16, 40, 6] is a method that uses bit vectors and benefits from the feature that the same bitwise operations (OR, AND, ADD, ... etc.) over groups of bits (or over individual bits) can be performed at once in parallel over the whole bit vector (see Figure 3.12). The representatives of the bit parallelism are Shift-Or, Shift-And, and Shift-Add algorithms.

The simulation using the bit parallelism will be shown in a Shift-Or algorithm that uses for each level (row) $l$, $0 \leq l \leq k$, of states one bit vector $R^l$ (of size $m$). Each state of the level then is represented by one bit in the vector. If a state is active,

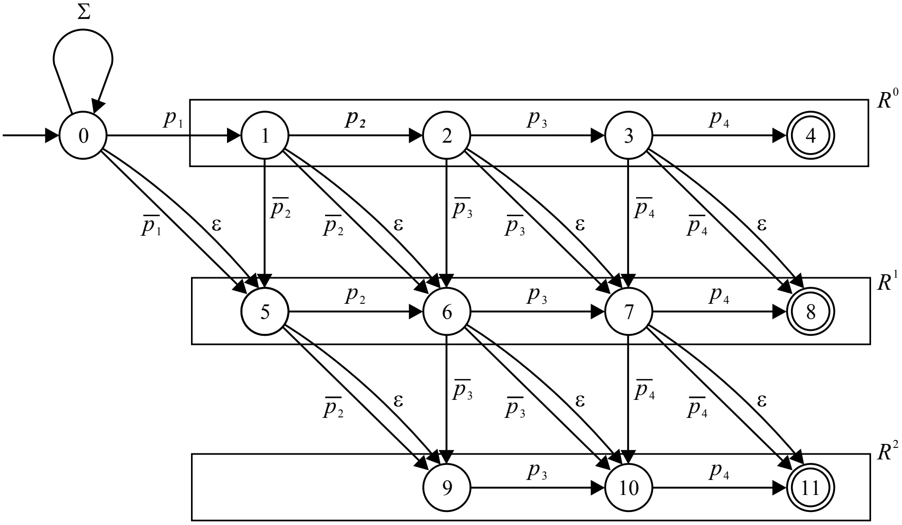


**Figure 3.12** Bit parallelism uses one bit vector $R$ for each level of states of NFA.

then the corresponding bit is 0; if it is not active, then the bit is 1. We have no bit representing $q_0$, because this state is always active. Formula (3.1) shows, how the vectors $R^l_i = [r^l_{1,i}, r^l_{1,i} \dots r^l_{m,i}]$ in the $i$th step are computed.

$$\begin{array}{rll}
r^l_{j,0} & \leftarrow 0, & 0 < j \le l, 0 \le l \le k \\
r^l_{j,0} & \leftarrow 1, & l < j \le m, 0 \le l \le k \\
R^0_i & \leftarrow \mathbf{shr}(R^0_{i-1}) \text{ or } D[t_i], & 0 < i \le n \\
R^l_i & \leftarrow (\mathbf{shr}(R^l_{i-1}) \text{ or } D[t_i]) & /\!* \text{ match } */ \\
 & \quad \text{and } (\mathbf{shr}(R^{l-1}_{i-1}) \text{ or not } D[t_i]) & /\!* \text{ replace } */ \\
 & \quad \text{and } \mathbf{shr}(R^{l-1}_i) & /\!* \text{ delete } */ \\
 & \quad \text{and } (R^{l-1}_{i-1} \text{ or not } \mathbf{shl}(D[t_i]) \text{ or } V), & /\!* \text{ insert } */ \\
 & & 0 < i \le n, 0 < l \le k
\end{array} \tag{3.1}$$

In the formula, **shr**() (respectively, **shl**()) is the bitwise operation right (respectively, left) shift, 'or' (respectively, 'and') is the bitwise operation OR (respectively, AND), and NOT() is a bitwise operation that exchanges 0s and 1s.

At the beginning, only the states of $\varepsilon$CLOSURE($\{q_0\}$) are active; therefore, the first $l$ bits are 0 in each vector $R^l_0$. The transitions representing matching are the only transitions leading to the states of level 0. To simulate these transitions, we shift vector $R^l_{i-1}$ to the right[4]—it is represented by term $\mathbf{shr}(R^l_{i-1})$. The operation **shr**() inserts 0 at the beginning of the vector, which implements the self-loop of the initial state. At this moment, all active states moved to the right, and we have to select only those transitions, which are labeled by $t_i$. For this selection, we use table $D$ of mask vectors. The table $D$ is defined as follows: $D[x] = [d_{1,x}d_{2,x} \dots d_{m,x}]$, $x \in \Sigma$, where $d_{j,x} = 0$, if $p_j = x$, or 1, otherwise. When we execute the bitwise operation or over the shifted vector and the mask vector $D[t_i]$, 1s is inserted in the positions where the transitions are not labeled by $t_i$. Thus, 0 remains only in such a position that the previous position contained 0, and there is a transition match labeled by $t_i$ connecting these two positions.

The term $\mathbf{shr}(R^{l-1}_{i-1})$ or not $((D[t_i]))$ represents *replace*. In this case, we shift the vector from the previous level $l - 1$. Then we have to select the transitions labeled by $t_i$. While in transition match, these transitions correspond to the positions matching the pattern; in case of transition *replace*, these transitions correspond to the positions mismatching the pattern. Therefore, we use mask vector not $(D[t_i])$.

The term $\mathbf{shr}(R^{l-1}_i)$ represents *delete*. In this case, we shift the new value of the vector from the previous level $l - 1$. We do not use any mask vector because we implement $\varepsilon$-transitions, which always apply. We use the new value of $R^{l-1}$ because no input symbol is read when $\varepsilon$-transitions are executed.

The term $R^{l-1}_{i-1}$ or not $(\mathbf{shl}(D[t_i]))$ or $V$ represents *insert*. We take the previous value of vector $R^{l-1}$. Because each such transition is labeled by $\overline{p}_j$ (*i.e.*, mismatching the label of the transition match leading from the same state), we use mask vector

[4] In practice, the Shift-Or algorithm has the right and left shifts exchanged because of easier implementation when the vectors are too long and have to be divided into several computer words.

**Table 3.2 Matrix *D* for pattern *p* = atggca**

| $D$ | $a$ | $g$ | $c$ | $t$ | $\Sigma \setminus \{a, g, c, t\}$ |
|---|---|---|---|---|---|
| $a$ | 0 | 1 | 1 | 1 | 1 |
| $t$ | 1 | 1 | 1 | 0 | 1 |
| $g$ | 1 | 0 | 1 | 1 | 1 |
| $g$ | 1 | 0 | 1 | 1 | 1 |
| $c$ | 1 | 1 | 0 | 1 | 1 |
| $a$ | 0 | 1 | 1 | 1 | 1 |

$D[t_i]$. We have to exchange 0 and 1 in the vector, but in addition, we have to shift this vector in the opposite direction than we usually shift the vectors. It is because we do not shift vector $R_{i-1}^{l-1}$ (it is a vertical transition that does not change a position in the pattern), we must get the corresponding bits to the correct positions in this way. Vector $V = [00\dots01]$ is used to prevent *insert* transitions leading into the final states.

All these terms are connected by bitwise operation and, which adds up all states arriving along all the transitions leading to the same state. If the last bit of vector $R^l$ is 0, then the final state of level $l$ is active. In this case, it means that the pattern has been found with at most $l$ differences.

**Example:**

Let $p =$ atggca, $t =$ atcagcaagatggca, and $k = 3$. Matrix $D$ is shown in Table 3.2, and Table 3.3 shows the simulation process of NFA run.

With the knowledge how bit parallelism simulates corresponding NFA, we could modify easily bit parallelism for other pattern matching tasks. One big advantage of bit parallelism is that we can define matching matrix $D$ for degenerate strings without changing the time and space complexity of the algorithm.

### 3.3.3 Dynamic Programming

The basic simulation method described in [19, 20] maintains a set of active states during the whole simulation process. Although in bit parallelism this set is represented by bit vectors, in dynamic programming, this set is represented by a vector of integer variables. We divide all states into some subsets, and each of the subsets is represented by one integer. The value of this integer then holds the information of what states of the subset are active.

The simulation using the dynamic programming will be shown on the NFA for the approximate string matching using the Levenshtein distance. This problem is defined as a searching for all occurrences of pattern $p = p_1p_2\dots p_m$ in text $t = t_1t_2\dots t_n$, where the found occurrence $x$ (substring of $t$) can have at most $k$ differences. The number of differences is given by the Levenshtein distance $D_L(p, x)$, which is defined as the minimum number of edit operations *replace*, *insert*, and *delete*, that are needed to convert $p$ to $x$.

**Table 3.3 Matrices $R^l$ for the approximate string matching using the Levenshtein distance ($p$ = atggca, $k$ = 3, and $t$ = atcagcaagatggca)**

| $R^0$ | - | $a$ | $t$ | $c$ | $a$ | $g$ | $c$ | $a$ | $a$ | $g$ | $a$ | $t$ | $g$ | $g$ | $c$ | $a$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a$ | 1 | **0** | 1 | 1 | **0** | 1 | 1 | **0** | **0** | 1 | **0** | 1 | 1 | 1 | 1 | **0** |
| $t$ | 1 | 1 | **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | 1 | 1 | 1 | 1 |
| $g$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | 1 | 1 | 1 |
| $g$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | 1 | 1 |
| $c$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | 1 |
| $a$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** |

| $R^1$ | - | $a$ | $t$ | $c$ | $a$ | $g$ | $c$ | $a$ | $a$ | $g$ | $a$ | $t$ | $g$ | $g$ | $c$ | $a$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a$ | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $t$ | 1 | **0** | **0** | **0** | **0** | **0** | 1 | **0** | **0** | **0** | **0** | **0** | 1 | 1 | 1 | **0** |
| $g$ | 1 | 1 | **0** | **0** | 1 | **0** | 1 | 1 | 1 | **0** | 1 | **0** | **0** | 1 | 1 | 1 |
| $g$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | 1 | 1 |
| $c$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** | 1 |
| $a$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | **0** |

| $R^2$ | - | $a$ | $t$ | $c$ | $a$ | $g$ | $c$ | $a$ | $a$ | $g$ | $a$ | $t$ | $g$ | $g$ | $c$ | $a$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a$ | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $t$ | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $g$ | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | 1 | **0** |
| $g$ | 1 | 1 | **0** | **0** | **0** | **0** | **0** | 1 | 1 | **0** | **0** | **0** | **0** | **0** | 1 | 1 |
| $c$ | 1 | 1 | 1 | **0** | 1 | 1 | **0** | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** | 1 |
| $a$ | 1 | 1 | 1 | 1 | **0** | 1 | 1 | **0** | 1 | 1 | 1 | 1 | 1 | **0** | **0** | **0** |

| $R^3$ | - | $a$ | $t$ | $c$ | $a$ | $g$ | $c$ | $a$ | $a$ | $g$ | $a$ | $t$ | $g$ | $g$ | $c$ | $a$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a$ | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $t$ | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $g$ | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $g$ | 1 | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| $c$ | 1 | 1 | **0** | **0** | **0** | **0** | **0** | **0** | 1 | **0** | **0** | **0** | **0** | **0** | **0** | 1 |
| $a$ | 1 | 1 | 1 | **0** | **0** | 1 | **0** | **0** | **0** | 1 | **0** | 1 | **0** | **0** | **0** | **0** |

Figure 3.13 shows the NFA constructed for this problem ($m = 4$, $k = 2$). The horizontal transitions represent matching, the vertical transitions represent *insert*, the diagonal $\varepsilon$-transitions represent *delete*, and the remaining diagonal transitions represent *replace*. The self-loop of the initial state provides skipping the prefixes of $t$ located in front of the occurrences. Formula (3.2) simulates the run of the NFA in Figure 3.13.

$$
\begin{array}{lll}
d_{j,0} \leftarrow j, & 0 \le j \le m & \\
d_{0,i} \leftarrow 0, & 0 \le i \le n & \\
d_{j,i} \leftarrow \textbf{min}(\textbf{if } t_i = p_j \textbf{ then } d_{j-1,i-1} \textbf{ else } d_{j-1,i-1} + 1, & & (3.2) \\
\qquad\qquad \textbf{if } j < m \textbf{ and } t_i \neq p_{j+1} \textbf{ then } d_{j,i-1} + 1, & 0 < i \le n, & \\
\qquad\qquad d_{j-1,i} + 1), & 0 < j \le m &
\end{array}
$$

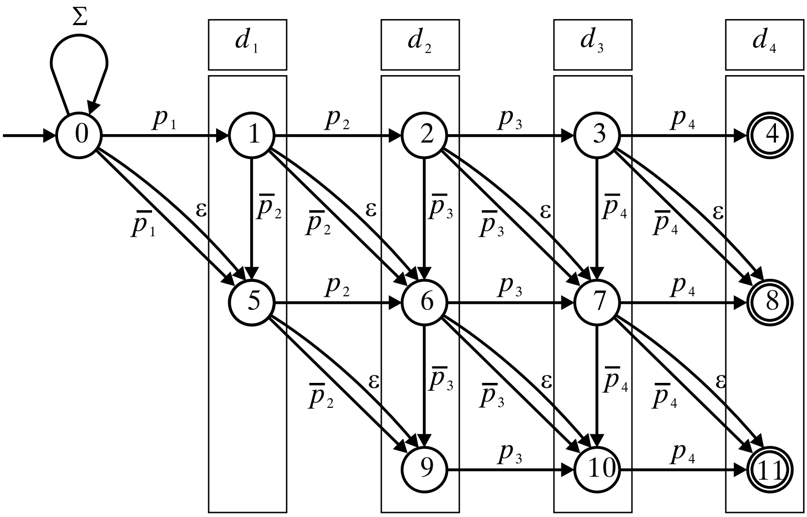


**Figure 3.13** Dynamic programming uses for each depth of states of NFA one integer variable *d*.

In the dynamic programming for the approximate string matching using the Levenshtein distance, there is for each depth $j$, $0 < j \leq m$, of NFA in each step $i$ of the run one integer variable $d_{j,i}$ that contains a level number of the topmost active state in $j$th depth of NFA. Each value of $d_{i,j}$ greater than $k + 1$ can be replaced by value $k + 1$ and represents that there is no active state in $j$th depth of NFA in $i$th step of the run.

The term $d_{j-1,i-1}$ represents matching transition, the term $d_{j-1,i-1} + 1$ represents transition *replace*, the term $d_{j,i-1} + 1$ represents transition *insert*, and the term $d_{j-1,i} + 1$ represents transition *delete*.

The self-loop of the initial state is represented by setting $d_{0,i} \leftarrow 0$, $0 \leq i \leq n$. Only the states reachable from $q_0$ by $\varepsilon$-transitions are active at the beginning. Thus, all transitions (paths) of the NFA are considered.

Each element $d_{m,i} \leq k$ shows an occurrence of $p$ with at most $d_{m,i}$ differences the final state in level $d_{m,i}$ is active. An example of matrix $D$ is shown in Table 3.4.

**Table 3.4 Matrix *D* for pattern *p* = adbbca and text *t* = adcabcaabadbbca using the Levenshtein distance**

| *D* | — | *a* | *d* | *c* | *a* | *b* | *c* | *a* | *a* | *b* | *a* | *d* | *b* | *b* | *c* | *a* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| — | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *a* | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| *d* | 2 | 1 | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 2 | 2 | 2 | 1 |
| *b* | 3 | 2 | 1 | 1 | 2 | 1 | 2 | 2 | 2 | 1 | 2 | 1 | 0 | 2 | 3 | 2 |
| *b* | 4 | 3 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | 1 | 0 | 3 | 3 |
| *c* | 5 | 4 | 3 | 2 | 3 | 3 | 2 | 3 | 4 | 3 | 3 | 3 | 2 | 1 | 0 | 4 |
| *a* | 6 | 5 | 4 | 3 | 2 | 4 | 3 | 2 | 3 | 4 | 3 | 4 | 3 | 2 | 1 | 0 |

The matrix $D$ can be compressed [17] using the property that $(d_{j,i} - d_{j-1,i-1}) \in \{0, 1\}$. The behavior of these simulation methods was studied with respect to changing the length of text, length of pattern, edit distance (Hamming, Levenshtein, and Damerau distances), and pattern class (English text, DNA sequence, image, ...).

### 3.3.4 Basic Simulation Method with Deterministic State Cache

There is a similarity between NFA simulation represented by BSM and NFA determinization. Each configuration of set $S$ of active states determines one state of the equivalent DFA. In fact, BSM creates states of DFA with three basic differences:

1. BSM creates states of DFA in the processing phase,[5] whereas NFA determinization creates them in the preprocessing phase.
2. BSM creates only those states of DFA it really needs, whereas NFA determinization creates all states of whole DFA.
3. BSM does not remember states of DFA. Once a new DFA states is computed, the previous one is forgotten.

That leads to an idea to create something in between. The BSM was extended by deterministic state cache (DSC) [22] where previously computed DFA states are stored. The only DFA states that are need are computed. If the transition needed during processing the input text is in DSC, then it is used; otherwise it is computed. If the DSC is full, then some cache technique to free some space in DSC are used. The BSM using DSC runs faster than original DSC but no faster than DFA.

The idea of on-the-fly construction of DFA originally was mentioned in [1], then it was discussed in [7] for the approximate string matching using Hamming distance, in [37] for the approximate string matching using Levenshtein distance and for the general problem in [31]. However, there was no practical implementation. In addition, [22] adds not only the practical implementation but also the cache of the deterministic states so that one can control the memory used.

## 3.4 FINITE AUTOMATON AS MODEL OF COMPUTATION

The finite automaton does not need to be run to solve some task. Even the construction of DFA can give us a solution. If we construct nondeterministic factor automaton, where each state represents one position in the input text (the position would be a number of the state), then by determinizing it by standard subset construction with preserving these positions (stored in d-subsets), we get deterministic factor automaton holding some extra information. For example, we easily can identify both exact and approximate repetitions in the input text as shown in [35].

[5] While working on the input text.

Another example is searching for all borders of the text by constructing an intersection of prefix and suffix automata as shown in [41]. The border of the text is a prefix, which is simultaneously a suffix of the text. As we know the automaton model behind the computation, we easily can extend into searching for the approximate borders as also shown in [41].

## 3.5 FINITE AUTOMATA COMPOSITION

The task of approximate string matching is to find out if a given text $t$ contains a pattern $p$ with some errors that are measured by an edit distance. Here, we decompose the problem into two subproblems:

- Approximate pattern matching for pattern $p$
- Searching for all factors of text $t$

We build an approximate pattern matching automaton for pattern $p$ (accepting language of all strings within a given edit distance from $p$) and a factor automaton for text $t$ (accepting all factors of $t$). The resulting solution is then an intersection automaton of the two automata (an automaton accepting intersection of the two languages). If we are interested only in the answer, then we do not need to construct whole intersection automaton. The algorithm drives the intersection computation in such a way so that the answer would be found as fast as possible. This approach then was used in [23, 28].

A more complicated composition is presented in [5] in which the finite automata approach was used to solve a DNA processing task. The task is to find common motifs with gaps in a set of input strings. The approaches used are the factor automaton, the computation of automaton accepting the union of languages of given automata, a subsequence automaton, and the computation of automaton accepting the intersection of languages of given automata presented already in [23]. Although other solutions of the problem require some limit of gaps (fixed gap, or bounded gap, or bounded sum of gaps), this algorithm allows any gaps while keeping the same time and space complexities like other algorithms requiring some limit of gaps.

## 3.6 SUMMARY

Table 3.5 summarizes the automata algorithms described in this chapter. It shows for each task (and method) whether pattern (P) and/or text (P) is preprocessed, the section describing the method, the preprocessing time, the running time, and the space required.

In the approximate pattern matching, the size of DFA ($|Q_{\mathrm{DFA}}|$) is $(k+1)!(k+2)^{m-2}$ [34]. The size $|Q_{\mathrm{NFA}}|$ of NFA for the approximate pattern matching is

**Table 3.5 Summary of automata algorithms**

| Task | P/T | Section | Preprocessing Time | Running Time | Space |
|---|---|---|---|---|---|
| Exact pattern matching (DFA) | P | 3.2.1 | $\mathcal{O}(m\lvert\Sigma\rvert)$ | $\mathcal{O}(n)$ | $\mathcal{O}(m\lvert\Sigma\rvert)$ |
| Exact pattern matching (SA) | T | 3.2.3 | $\mathcal{O}(n)$ | $\mathcal{O}(m)$ | $\mathcal{O}(n)$ |
| Exact pattern matching (KMP) | P | 3.2.6 | $\mathcal{O}(m)$ | $\mathcal{O}(n)$ | $\mathcal{O}(m)$ |
| Exact pattern matching (BMH) | P | 3.2.5 | $\mathcal{O}(m)$ | $\mathcal{O}(nm)^*$ | $\mathcal{O}(m)$ |
| Exact pattern matching (BNDM) | P | 3.2.5 | $\mathcal{O}(m)$ | $\mathcal{O}(nm)^*$ | $\mathcal{O}(m)$ |
| Exact pattern matching (BOM) | P | 3.2.5 | $\mathcal{O}(m)$ | $\mathcal{O}(nm)^*$ | $\mathcal{O}(m)$ |
| Approximate pattern matching (DFA) | P | 3.2.1 | $\mathcal{O}(\lvert Q_{\mathrm{DFA}}\rvert\lvert Q_{\mathrm{NFA}}\rvert\lvert\Sigma\rvert)$ | $\mathcal{O}(n)$ | $\mathcal{O}(\lvert Q_{\mathrm{DFA}}\rvert.\lvert\Sigma\rvert)$ |
| Approximate pattern matching (BSM) | P | 3.3.1 | $\mathcal{O}(\lvert Q_{\mathrm{NFA}}\rvert\lvert\Sigma\rvert\lceil\frac{\lvert Q_{\mathrm{NFA}}\rvert}{w}\rceil)$ | $\mathcal{O}(n\lvert Q_{\mathrm{NFA}}\rvert\lceil\frac{\lvert Q_{\mathrm{NFA}}\rvert}{w}\rceil)$ | $\mathcal{O}(\lvert Q_{\mathrm{NFA}}\rvert\lvert\Sigma\rvert\lceil\frac{\lvert Q_{\mathrm{NFA}}\rvert}{w}\rceil)$ |
| Approximate pattern matching (BP) | P | 3.3.2 | $\mathcal{O}(\lvert\Sigma\rvert\lceil\frac{m}{w}\rceil+m)$ | $\mathcal{O}(nk\lceil\frac{m}{w}\rceil)$ | $\mathcal{O}((\lvert\Sigma\rvert+k+1)\lceil\frac{m}{w}\rceil)$ |
| Approximate pattern matching (DP) | P | 3.3.3 | $\mathcal{O}(m)$ | $\mathcal{O}(nm)$ | $\mathcal{O}(m)$ |
| Approximate pattern matching (automata intersection) | T, P | 3.5 | $\mathcal{O}(\lvert Q_{\mathrm{NFA}}\rvert+n)$ | $\mathcal{O}((k+1)!(k+2)^{m-k+1})$ | $\mathcal{O}(\lvert Q_{\mathrm{NFA}}\rvert+n)$ |

$(m+1)(k+1) - \frac{k(k+1)}{2}$. Machine word size $w$ plays an important role in bit parallelism and bitwise implementation of BSM. If the bit vectors are longer than $w$, then they have to be divided into several machine words. BMH, BNDM and BOM (marked by $*$) run in $\mathcal{O}(nm)$ in the worst case but in sublinear time on average. Note that the running time of the approximate string matching using automata intersection does not depend on the length of input text.

## REFERENCES

1. A.V. Aho. Pattern matching in strings. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*. Academic Press, London, U.K., 1980, pp. 325–347.
2. A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun ACM*, 18(6):333–340, 1975.
3. A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers—Principles, Techniques and Tools*. Addison-Wesley, Reading, MA, 1986.
4. C. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle: A new structure for pattern matching. In J. Pavelka, G. Tel, and M. Bartošek, editors, *SOFSEM'99, Theory and Practice of Informatics*, number 1725. Lecture Notes in Computer Science, Milovy, Czech Republic, 1999. Springer-Verlag, Berlin, Germany, pp. 291–306.
5. P. Antoniou, J. Holub, C.S. Iliopoulos, B. Melichar, and P. Peterlongo. Finding common motifs with gaps using finite automata. In O. H. Ibarra and H.-C. Yen, editors, *Implementation and Application of Automata*, number 4094. Lecture Notes in Computer Science. 2006, Springer-Verlag, Heidelberg, Germany, pp. 69–77.
6. R.A. Baeza-Yates and G.H. Gonnet. A new approach to text searching. *Commun ACM*, 35(10):74–82, 1992.
7. R.A. Baeza-Yates and G.H. Gonnet. Fast string matching with mismatches. *Inf Comput*, 108(2):187–199, 1994.
8. M. Balík. DAWG versus suffix array. In J.-M. Champarnaud and D. Maurel, editors, *Implementation and Application of Automata*, number 2608. Lecture Notes in Computer Science. 2003, Springer-Verlag, Heidelberg, Germany, pp. 233–238.
9. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theor Comput Sci*, 40(1):31–55, 1985.
10. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnel. Complete inverted files for efficient text retrieval and analysis. *J Assoc Comput Mach*, 34(3):578–595, 1987.
11. R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Commun ACM*, 20(10): 762–772, 1977.
12. M. Crochemore and C. Hancart. Automata for matching patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2 Linear Modeling: Background and Application. Springer-Verlag, Berlin, Germany, pp. 399–462, 1997.
13. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.

14. M. Crochemore and R. Vérin. Direct construction of compact directed acyclic word graphs. In A. Apostolico and J. Hein, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, number 1264 Lecture Notes in Computer Science, Aarhus, Denmark, 1997. Springer-Verlag, Berlin, Germany, pp. 116–129.
15. F. Damerau. A technique for computer detection and correction of spelling errors. *Commun ACM*, 7(3):171–176, 1964.
16. B. Dömölki. An algorithm for syntactical analysis. *Comput Ling*, 3:29–46, 1964. Hungarian Academy of Science, Budapest.
17. Z. Galil and K. Park. An improved algorithm for approximate string matching. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, number 372 Lecture Notes in Computer Science, Stresa, Italy, 1989. Springer-Verlag, Berlin, Germany, pp. 394–404.
18. R.W. Hamming. Error detecting and error correcting codes. *Bell Syst Tech J*, 29(2):147–160, 1950.
19. J. Holub. *Simulation of Nondeterministic Finite Automata in Pattern Matching*. PhD Dissertation, Czech Technical University in Prague, Czech Republic, February 2000.
20. J. Holub. Bit parallelism—NFA simulation. In B.W. Watson and D. Wood, editors, *Implementation and Application of Automata*, number 2494 Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 2002, pp. 149–160.
21. J. Holub and M. Crochemore. On the implementation of compact DAWG's. In J.-M. Champarnaud and D. Maurel, editors, *Implementation and Application of Automata*, number 2608 Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 2003, pp. 289–294.
22. J. Holub and T. Kadlec. NFA simulation using deterministic state cache. In J. Chan, J. W. Daykin, and M. S. Rahman, editors, *London Algorithmics 2008: Theory and Practice*. College Publications, 2009. To appear.
23. J. Holub and B. Melichar. Approximate string matching using factor automata. *Theor Comput Sci*, 249(2):305–311, 2000.
24. J. Holub, W.F. Smyth, and S. Wang. Hybrid pattern-matching algorithms on indeterminate strings. In J. Daykin, M. Mohamed, and K. Steinhoefel, editors, *London Stringology Day + London Algorithmic Workshop 2006*, King's College London Series Texts in Algorithmics, 2007, pp. 115–133.
25. J. Holub and P. Špiller. Practical experiments with NFA simulation. In L. Cleophas and B. W. Watson, editors, *Proceedings of the Eindhoven FASTAR Days 2004*, TU Eindhoven, The Netherlands, 2004, pp. 73–95.
26. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata, Languages and Computations*. Addison-Wesley, Reading, MA, 1979.
27. R.N. Horspool. Practical fast searching in strings. *Softw—Pract Exp*, 10(6):501–506, 1980.
28. S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi. On-line construction of compact directed acyclic word graphs. *J Discrete Algorithm*, 146(2):156–179, 2005.
29. D.E. Knuth, J.H. Morris, Jr, and V.R. Pratt. Fast pattern matching in strings. *SIAM J Comput*, 6(2):323–350, 1977.
30. D.C. Kozen. *Automata and Computability*. Springer-Verlag, Berlin, Germany, 1997.

31. S. Kurtz. *Fundamental Algorithms for a Declarative Pattern Matching System*. PhD Dissertation, Technische Fakultt, Universitt Bielefeld, September 1995. Available as Report 95–03.
32. S. Kurtz. Reducing the space requirements of suffix trees. *Softw—Pract Exp*, 29(13):1149–1171, 1999.
33. V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Sov Phys Dokl*, 6:707–710, 1966.
34. B. Melichar. Space complexity of linear time approximate string matching. In J. Holub, editor, *Proceedings of the Prague Stringologic Club Workshop '96*, Czech Technical University in Prague, Czech Republic, 1996. Collaborative Report DC–96–10, pp. 28–36.
35. B. Melichar. Repetitions in text and finite automata. In L. Cleophas and B. W. Watson, editors, *Proceedings of the Eindhoven FASTAR Days 2004*, TU Eindhoven, The Netherlands, 2004, pp. 1–46.
36. J.H. Morris, Jr and V.R. Pratt. A linear pattern-matching algorithm. Report 40, University of California, Berkeley, CA, 1970.
37. G. Navarro. A partial deterministic automaton for approximate string matching. In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing*, Carleton University Press, Valparaiso, Chile, 1997, pp. 95–111.
38. G. Navarro and M. Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In M. Farach-Colton, editor, *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, number 1448 Lecture Notes in Computer Science, Piscataway, NJ, 1998. Springer-Verlag, Berlin, Germany, pp. 14–33.
39. Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Nomenclature for incompletely specified bases in nucleic acid sequences. recommendations 1984. *Biochem J*, 229:281–286, 1985.
40. R. K. Shyamasundar. A simple string matching algorithm. Technical report, Tata Institute of Fundamental Research, India, 1976.
41. M. Šimůnek and B. Melichar. Borders and finite automata. In O. H. Ibarra and H.-C. Yen, editors, *Implementation and Application of Automata*, number 4094 Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 2006, pp. 58–68.
42. D.M. Sunday. A very fast substring search algorithm. *Commun ACM*, 33(8):132–142, 1990.
43. K. Thompson. Regular expression search algorithm. *Commun ACM*, 11:419–422, 1968.

# 4

# NEW DEVELOPMENTS IN PROCESSING OF DEGENERATE SEQUENCES

Pavlos Antoniou and Costas S. Iliopoulos

## 4.1 INTRODUCTION

Degenerate sequences are sequences that have several possible letters in some of their positions. In terms of biological sequences, degenerate sequences can have more than one base or amino acid in some positions. For example, in the DNA sequence AG[CT]ACC[ACT]A, at position 3, we have either C or T, and in position 7 we can have either A, C, or T.

The processing of these degenerate sequences presents problems that have interested researchers because of their direct applications in biology, cryptography, and music. In music, for example, single nodes may match chords. In cryptography, undecoded symbols may match one of a specific set of letters in the alphabet [9].

In computational biology research, degenerate sequences have been used extensively to represent polymorphisms in DNA/RNA sequences. These polymorphisms in coding regions are caused by redundancy of the genetic code or polymorphism in binding sites or plainly by errors and limitations of the sequencing equipment in biological labs. Additionally, biologists have been interested in degenerate sequences especially for the problem of degenerate primer design in polymerase chain reaction (PCR) sequences [17] .

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

### 4.1.1 Degenerate Primer Design Problem

PCR, is a process that amplifies a specific region of DNA to provide enough copies of that region to be tested or sequenced. To use this PCR process, the biologists need to know the exact sequences, which lie on either side of the region of interest. These sequences are used to design two synthetic DNA oligonucleotides and are called *primers*. Primers usually have a length around 20–30 base pairs [14].

The PCR primer sequences are often degenerate as some of their positions have several possible bases. The *degeneracy* of a primer is the number of unique sequence combinations it contains [14]. The primer design problem usually involves a set of sequences for which we wish to design primers that can match as many of the sequences of the set as possible. For example, the primer $P = TT[CG]C[ACT]G$ covers all four of the following strings:

$$
\begin{aligned}
S_1 &= TTGCAG\\
S_2 &= TTCCAG\\
S_3 &= TTGCTG\\
S_4 &= TTGCCG
\end{aligned}
$$

To find solutions for these problems associated with degenerate sequences, the repetitive structures and properties that can be found in degenerate strings have been the subject of research in string algorithms. In recent years, there were many algorithms concerning degenerate strings, and the advances in this area provided solutions in cryptography, music, and biology.

In this chapter, we will investigate and present new algorithms to find repetitive structures in degenerate strings. We begin with the problem of finding local and global covers and repetitive structures called seeds in the degenerate strings. We present an algorithm for finding the smallest cover of the string $x$ in $O(n \log n)$ time, where $n$ is the length of the string $n$. Subsequently, we extend this algorithm to find all local and global covers of the string $x$ and we extend the latter to compute the seeds of the string.

Subsequently, we study the problem of finding local and global covers as well as seeds in *conservative* degenerate strings. A conservative degenerate string is a degenerate string in which the number of degenerate symbols in the positions of the string (*i.e.*, the nonsolid symbols), is bounded by a constant $\kappa$. We present an algorithm for finding a conservative degenerate pattern $p$ in a degenerate string $x$. Furthermore, we present algorithms for computing conservative covers and seeds of a degenerate string $x$.

## 4.2 BACKGROUND

As mentioned, in this chapter, we will investigate algorithms that find repetitive structures in degenerate sequences. These structures include covers and seeds of the strings.

Covers are considered common regularities in a string along with repetitions and periods. They are periodically repetitive. A substring $w$ of a string $x$ is called a cover of $x$ if and only if $x$ can be constructed by concatenations and superpositions of $w$. A seed is an extended cover in the sense of a cover of a superstring of $x$.

Finding the regularities present in strings is not only interesting in string algorithms, but it is also useful in many applications. These applications include molecular biology, data compression, and computational music analysis. Regularities in strings have been studied widely in the last 20 years. There are several $O(n \log n)$ time algorithms for finding repetitions [6], in a string $x$, where $n$ is the length of $x$. Apostolico and Breslauer [2] gave an optimal $O(\log \log n)$ time parallel algorithm for finding all repetitions. The preprocessing of the Knuth–Morris–Pratt algorithm [13] finds all periods of every prefix of $x$ in linear time.

In many cases, it is desirable to relax the meaning of repetition. For instance, if we allow overlapping and concatenations of periods in a string, then we get the notion of *covers*. The notion of covers was introduced by Apostolico, Farach, and Iliopoulos in [3] in which a linear-time algorithm to test superprimitivity was given. Moore and Smyth in [15] gave linear-time algorithms for finding all covers of a string $x$.

An extension of the notion of covers, is that of *seeds* (*i.e.*, covers of a superstring of $x$). The notion of seeds was introduced by Iliopoulos, Moore, and Park [11] and an $O(n \log n)$ time algorithm was given for computing all seeds of $x$. A parallel algorithm for finding all seeds was presented by Berkman, Iliopoulos, and Park [5], that requires $O(\log n)$ time and $O(n \log n)$ work.

In this chapter, we find these string regularities in degenerate strings. Figure 4.1 presents an example of a degenerate biological sequence in which in some positions, we do not have only one base but may have up to four. That means that in those particular positions, it is not clear which base resides there; in other words the base in those positions is not *determined.*

An algorithm was described [8] for computing all occurrences of a pattern $p$ in a text string $x$, where both $p$ and $x$ are defined on the alphabet $\Sigma^*$, but although efficient in theory, the algorithm was not useful in practice. Indeterminate string pattern matching mainly has been handled by bit mapping techniques (ShiftOr method) [4],[19]. These techniques have been used to find matches for a degenerate pattern $p$ in a string $x$ [9] and the agrep utility [18] has been virtually one of the few practical algorithms available for degenerate pattern matching.

In [9], the authors extended the notion of degenerate strings by distinguishing two distinct forms of degenerate match ("quantum" and "deterministic"). Roughly speaking, a "quantum" match allows a degenerate letter to match two or more distinct letters during a single matching process; a "determinate" match restricts each degenerate letter to a single match [9].

In the area of biology, we can find the notion of *conservative* degenerate strings. In biology, usually, the number of degenerate positions in a sequence is bounded naturally by a constant value. Otherwise, we would have a cover of length 1 with just a do not care symbol that corresponds to all letters of the alphabet $\Sigma$. Therefore, we impose a constraint on the strings, which requires that the number of degenerate positions in a cover $c$ is less than the constant, that is, a "conservative" cover. An example

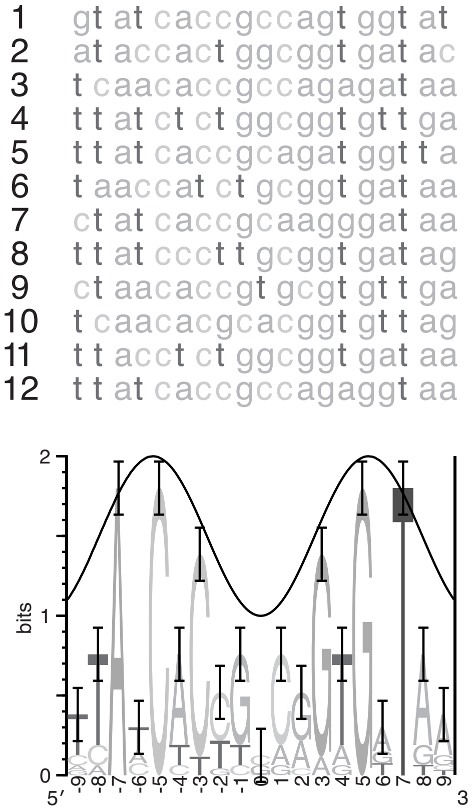


**Figure 4.1** A sequence logo of a biological degenerate sequence. Picture taken from [16].

of a sequence containing degenerate positions is shown in Figure 4.1, which depicts a sequence logo of a degenerate sequence. The bottom logo is the consensus sequence derived by the 12 sequences on top of it. If we look at the logo, then we can see that position 1 is degenerate as we can have [TCAG] occurring, position 2 is degenerate also having a possible occurrence of [TCA], position 3 is solid, nondegenerate, as in that position, only A occurs.

This chapter is organized as follows. Section 4.3 presents the basic definitions and notations used throughout this chapter. Section 4.4 presents linear-time algorithms to find covers and seeds in degenerate strings, and section 4.5 presents algorithms for finding repetitive structures in linear time for conservative degenerate strings. Finally, section 4.6 presents some concluding remarks.

## 4.3 BASIC DEFINITIONS

A *string* is a sequence of zero or more symbols from the alphabet $\Sigma$. The set of all strings over $\Sigma$ is denoted by $\Sigma^*$. The length of a string $x$ is denoted by $|x|$. The *empty* string, the string of length zero, is denoted by $\epsilon$. The $i$-th symbol of a string $x$ is denoted by $x[i]$.

A string $w$ is a substring of $x$ if $x = uwv$, where $u, v \epsilon \Sigma^*$. We denote by $x[i \ldots j]$ the substring of $x$ that starts at position $i$ and ends at position $j$. Conversely, $x$ is called a *superstring* of $w$. A string $w$ is a *prefix* of $x$ if $x = wy$ for $y \epsilon \Sigma^*$. Similarly, $w$ is a *suffix* of $x$ if $x = yw$ for $y \epsilon \Sigma^*$.

We call a string $w$ a *subsequence* of $x$ (or $x$ is a supersequence of $w$) if $w$ is obtained by deleting zero or more symbols at any positions from $x$. For example, $ace$ is a subsequence of $aabcdef$. For a given set $S$ of strings, a string $w$ is called a common supersequence of $S$ if $w$ is a supersequence of every string in $S$.

The string $xy$ is the *concatenation* of the strings $x$ and $y$. The concatenation of $k$ copies of $x$ is denoted by $x^k$. For two strings $x = x[1 \ldots n]$ and $y = y[1 \ldots m]$ such that $x[n - i + 1 \ldots n] = y[1 \ldots i]$ for some $i \geq 1$ (*i.e.*, such that $x$ has a suffix equal to a prefix of $y$), the string $x[1 \ldots n]y[i + 1 \ldots m]$ is a *superposition* of $x$ and $y$. We also say that $x$ overlaps with $y$. A substring $y$ of $x$ is called a *repetition* in $x$ if $x = uy^kv$, where $u, y, v$ are substrings of $x$ and $k \geq 2$, $|y| \neq 0$. For example, if $x = aababab$, then $a$ (appearing in positions 1 and 2) and $ab$ (appearing in positions 2, 4, and 6) are repetitions in $x$; in particular, $a^2 = aa$ is called a *square*, and $(ab)^3 = ababab$ is called a *cube*.

A nonempty substring $w$ is called a *period* of a string $x$, if $x$ can be written as $x = w^kw'$ where $k \leq 1$ and $w'$ is a prefix of $w$. The shortest string of the periods of $x$ is called the period of $x$. For example, if $x = abcabcab$, then $abc$, $abcabc$, and the string $x$ itself are periods of $x$, whereas $abc$ is the period of $x$.

A substring $w$ of $x$ is called a *cover* of $x$, if $x$ can be constructed by concatenating or overlapping copies of $w$. We also say that $w$ *covers* $x$. For example, if $x = ababaaba$, then $aba$ and $x$ are covers of $x$. If $x$ has a cover $w \neq x$, then $x$ is *quasiperiodic*; otherwise, $x$ is *superprimitive*.

A substring $w$ of $x$ is called a *seed* of $x$ if $w$ covers one superstring of $x$ (this can be any superstring of $x$, including $x$ itself). For example, $aba$ and $ababa$ are some seeds of $x = ababaab$.

A *degenerate string* is a sequence $T = T[1]T[2] \ldots T[n]$, where $T[i] \subseteq \Sigma$ for each $i$, and $\Sigma$ is a given alphabet of a potentially large size. When a position of the string is degenerate, and it can match more than one element from the alphabet $\Sigma$, we say that this position has *nonsolid* symbol. If in a position, only one element of the alphabet $\Sigma$ is present, then we refer to this symbol as *solid*. A *conservative* degenerate string, is a degenerate string in which its number of degenerate symbols is bounded by a constant $k$.

***Regular Expressions and Languages.*** Any subset of $\Sigma^*$ is a language on the alphabet $\Sigma$. The regular expressions on an alphabet $\Sigma$ and the regular languages they describe are defined recursively as follows [7]:

1. 0 and 1 are regular expressions that recursively describe the empty set $\emptyset$ and$\{\epsilon\}$,

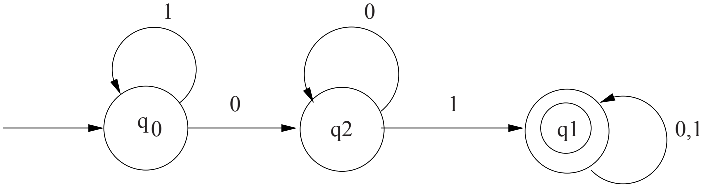


**Figure 4.2** The transition diagram of a DFA accepting all strings that have a substring 01.

2. For every letter $a \in \Sigma$, $a$ is a regular expression that describes the singleton $\{a\}$,
3. If $x$ and $y$ are regular expressions, respectively describing the regular languages $X$ and $Y$, then $(x) + (y)$, $(x) \times (y)$, and $(x)^*$ are regular expressions that describe the regular languages $X \cup Y$, $X \times Y$, $X^*$, respectively.

A deterministic finite automaton (DFA), consists of the following [10]:

1. A finite set of states denoted $\mathcal{Q}$
2. A finite set of input symbols often denoted $\Sigma$
3. A transition function denoted $\delta$ that takes as an argument a state and an input symbol and returns a state
4. A start state one state of $\mathcal{Q}$
5. A set of final states $\mathcal{F}$, where $\mathcal{F}$ is a subset of $\mathcal{Q}$.

A DFA can be represented as a five-tuple notation $A = (\mathcal{Q}, \Sigma, \delta, q_0, F)$, where $A$ is the name of the DFA, $\mathcal{Q}$ is its set of states, $\Sigma$ is its set of input symbols, $\delta$ is its transition function, $q_0$ is its starting state, and $\mathcal{F}$ is its set of final states [10].

An example of a DFA is presented in Figure 4.2, which presents the transition diagram of a DFA accepting all strings with a substring 01.

A nondeterministic finite automaton (NFA), has the ability to be in several states at once, whereas the DFA for every pair of states and transition function exists at most one receiving state exists [10].

***The Aho-Corasick Automaton [1].*** The Aho-Corasick Automaton for a given finite set $P$ of patterns is a deterministic finite automaton $G$ accepting the sets of all words containing a word of $P$ as a suffix.

$G = (Q, \Sigma, g, f, q_0, F)$, where function $Q$ is the set of states, $\Sigma$ is the alphabet, $g$ is the forward transition, $f$ is the failure link (*i.e.*, $f(q_i) = q_j$), if and only if $S_j$ is the longest suffix of $S_i$ that is also a prefix of any pattern, $q_0$ is the initial state, and $F$ is the set of final (terminal) states [1]. The construction of the Aho Corasick (AC) automaton can be done in $O(d)$-time and space complexity, where $d$ is the size of the dictionary (*i.e.*, the sum of the lengths of the patterns that the AC automata will match).

**Theorem 4.1** ([12]) *Let $a[1], \ldots, a[n]$ be a doubly linked list. An algorithm exists that preprocess the list a in such way that after several deletions in the list a, one can find the nearest $a[j]$ to the left of $a[i]$ with $a[j] \leq a[i]$ in constant time.*

## 4.4 REPETITIVE STRUCTURES IN DEGENERATE STRINGS

### 4.4.1 Using the Masking Technique

To match efficiently character classes, we represent our strings as a sequence of four bit masks. The alphabet for describing DNA sequences has four symbols, namely {A, C, G, T}. We convert these single characters to represent the set of bit masks {1000, 0100, 0010, 0001}.

For $k$ characters, $x_1 \ldots x_k$, we can represent the character set $[x_1 \ldots x_k]$ as follows:

$$M(x_1) \text{ OR } \mathrm{M(x_2)} \text{ OR } \ldots \text{ OR } \mathrm{M(x_k)}$$

where $M(x_i)$ is the 4 bit mask of $x_i$. Using this representation and the bitwise AND operation, we can determine whether there is a match between characters or character sets. Where a nonzero result would indicate a match, and a zero result would indicate a mismatch.

For example, if we wanted to determine whether [AC] matched with [CG], then we first would convert the character sets into the following four bit masks: [AC] $=$ 1000 OR 0100 $=$ 1100, [CG] $=$ 0100 OR 0010 $=$ 0110. We then perform a bitwise AND operation on the four bit masks:1100 AND 0110 $=$ 0100. Because we have a nonzero result, we can conclude [AC] matched with [CG], as they have C as a common symbol (the character representation of the resulting bit mask).

In the algorithms of this chapter, we will be applying bit masking to the degenerate strings.

### 4.4.2 Computing the Smallest Cover of the Degenerate String *x*

The following algorithm finds the smallest cover $\hat{u}$ that covers the degenerate string $x$. Assume that we have performed $k$-iterations. So far, we have built the following:

***Position Array.*** We find all occurrences of substring $\hat{u}$ in $x$, and we denote the occurrence of $\hat{u}$ at position $I_\mu$ as $u^\mu$. Then $u^1$ is a prefix of $x$ and $|\hat{u}| = k$. Then the cover $\hat{u} = u^\mu$, $1 \leq \mu \leq \ell$. The starting positions of each substring $u$ is noted in the following array $S$:

$$S_1 = \{I_1, I_2, \ldots I_{\mu-1}, I_\mu, I_{\mu+1} \ldots I_\ell\}$$

***Gap Array.*** The distances between the starting positions of consecutive $u^\mu$, $u^{\mu+1}$ is denoted by $g_i$ and $i$ is entered into a second array $S_2$. Then the array $S_2$ is as follows:

$$S_2 = \{g_1, g_2, \ldots g_{\mu-1}, g_\mu, g_{\mu+1} \ldots g_\ell\}$$

Figure 4.3 presents the distances $g_i$ between the substrings $u$ as arcs between the substrings.

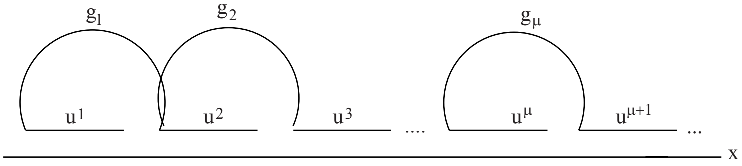


**Figure 4.3** Covering the string *x* with substring *u*.

***Order Array.*** We create the array $S_3$, which holds the elements of array $S_2$ sorted from smallest to largest. We also create a doubly linked list $L$, where for each element $g_i$ in $S_2$, we keep its Order $(g_i)$ according to $S_3$. The reason we create this doubly linked list is to be able to use Theorem 1. This theorem allows us to access any element of $S_3$ in constant time.

***Testing.*** We apply a simple test to determine whether the substring $\hat{u}$ is a cover of $x$. We check whether the largest element of $S_3$, $g_\ell$ is smaller than $k$. If $g_\ell \leq k$, then $\hat{u}$ is a cover of $x$.

If $\hat{u}$ is not a cover of $x$ for $|\hat{u}| = k$, then we continue by extending $k$ by 1 and solve the problem for $|u| = k + 1$. Accordingly, the distance allowed between the substrings for them to be considered as covers also is increased to $k + 1$.

***Main Steps.*** We extend the length of each $u^\mu$, $1 \leq \mu \leq \ell$ by one character, to length $k + 1$. So far, we have a series of prefixes of length $k$ to check whether they can be extended by one character. Let $u^i_{k+1}$ denote the $k + 1$-th character of $u^i$.

Step 4.1

We check to see whether the next character of the current substring is equal to the next character of another substring (*i.e.*, if $u^i_{k+1}$ is equal to $u^j_{k+1}$). This check can be performed via the bit masking method for index $i$, which is a good method for practical purposes without affecting the running time of the algorithm.

Let

$$u^{i_1}_{k+1} = u^{i_2}_{k+1} = \ldots = u^{i_\tau}_{k+1}$$

and let their starting positions

$$I = \{i_1, i_2 \ldots i_\tau\}$$

Then, $\hat{u_k} = \{a | a \in u^i_{k+1}, i \in I\}$ and $\hat{u} = \hat{u_1}\hat{u_2} \ldots \hat{u_k}$.

Step 4.2

Suppose in one position $I_\mu$, $u_\mu$ cannot be extended further without giving a mismatch. This is illustrated in Figure 4.4 at the point marked by ×. But matching substrings follow this unmatched substring. Therefore, we want to discard this $u_\mu$ and

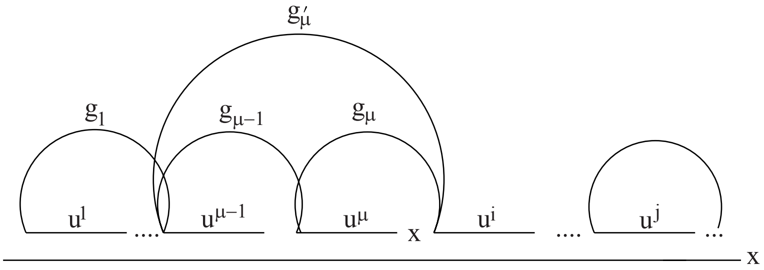


**Figure 4.4** Extending the length of substring *u* from *k* to *k*+ 1 to find a cover for string *x*. At the position marked by a cross, substring *u* cannot be extended further without giving a mismatch.

cover the string with the rest of the substrings $u_i$. We do this by first deleting $I_\mu$ from $S_1$. Then we delete and update the distances between $g_{\mu-1}$ and $g_{\mu+1}$ in $S_2$.

$$S_1 = \{I_1, I_2, \dots I_{\mu-1}, \cancel{I_\mu}, I_{\mu+1} \dots I_\ell\}$$
$$S_2 = \{g_1, g_2, \dots g_{\mu-1}, \cancel{g_\mu}, g_{\mu+1} \dots g_\ell\}$$

By keeping the linked list $L$ and from Theorem 1, the corresponding distances $g_{\mu-1}$ and $g_{\mu+1}$ in $S_2$ can be found in constant time.

STEP 4.3
We do a binary search and insert the distance $g_{\mu'}$ in its corresponding position in the sorted set $S_2$. This requires $O(\log n)$ time.

We test whether $|g_\ell| \leq k+1$. If this equation is true, then $\hat{u}$ with $|\hat{u}| = k+1$ is a cover of $x$.

Figure 4.4, shows an example of this operation. Arcs $g_{\mu-1}$ and $g_\mu$ will be deleted and will be replaced by $g_\mu{}'$.

### 4.4.3 Computing Maximal Local Covers of *x*

The following algorithm finds maximal substrings of $x$, which are covered locally by some nonextendable factor, $\hat{u}$ of $x$. As with Algorithm 4.4.2, we assume that we have performed $k$-iterations. However, the algorithm varies in that we now not only are concerned with $u$ that is a prefix of $x$. We therefore are considering all factors of $x$ (starting with length two) as possible local covers of $x$. After $k$-iterations, we would have created the following:

***Position Array.*** We have a set of local covers $\{\hat{u}^{(1)}, \hat{u}^{(2)}, \dots, \hat{u}^{(\lambda)}\}$ with $|\hat{u}_i| = k$, $1 \leq i \leq \lambda$. The starting positions of each substring $\hat{u_j}$ is noted in an array $S_1^{(j)}$.

$$S_1^{(j)} = \{I_1^{(j)}, I_2^{(j)}, \dots I_{\mu-1}^{(j)}\}$$

We perform the following steps for each $j$, $1 \leq j \leq \lambda$:

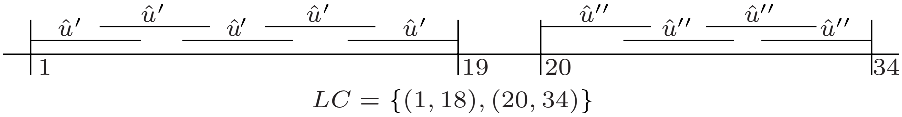


**Figure 4.5** Example of LC array, supposing that $\hat{u}$ and $\hat{u}'$ are local covers.

***Gap Array.*** The distances between the starting positions of consecutive $\hat{u}^{(j)}$ substrings are entered into a second array $S_2^{(j)}$. We denote the distances between consecutive $u_i^{(j)}$-th and $u_{i+1}^{(j)}$-th occurrences of $\hat{u}^{(j)}$ as $g_i^{(j)}$; then the array $S_2^{(j)}$ is as follows:

$$S_2^{(j)} = \{g_1^{(j)}, g_2^{(j)}, \dots g_\mu^{(j)}, \dots g_\ell^{(j)}\}$$

***Order Array.*** We create array $S_3^{(j)}$, which holds the elements of array $S_2^{(j)}$ sorted from smallest to largest. We also create a doubly linked list $L$, where for each element $g_i^{(j)}$ in $S_2^{(j)}$, and we keep its Order $(g_i^{(j)})$ according to $S_3^{(j)}$. As with Algorithm 4.4.2, the doubly linked list is needed to use Theorem 1.

***Local Covers.*** We create an array LC of local covers that have been detected up to now in the algorithm. Each cover is stored as a set of pairs, $(\Lambda_l^{(j)}, \Lambda_r^{(j)})$, where $\Lambda_l^{(j)}$ and $\Lambda_r^{(j)}$ are the left-most and right-most positions of the $i$-th local cover of $x$, respectively. Figure 4.5 shows an example of the local covers array.

***Main Steps.*** We extend the length of each $\hat{u}_j$ by one character to length $k+1$. We then partition the set $S_1^{(j)}$ into sets to represent all possible extensions $\hat{u}_j$ of length $k+1$. The following steps are repeated until $\hat{u}$ cannot be extended any further. Figure 4.6 presents the extension for local covers, and Figure 4.7 presents the partition of the set $S$.

STEP 4.4
We check to see whether the next character of the current substring is equal to the next character of another substring (*i.e.*, if $u_{k+1}^{i,(j)}$ is equal to $u_{k+1}^{v,j}$). This check can be performed via the bit masking method for index $i$, which is a good method for practical purposes without affecting the running time of the algorithm.

Let

$$u_{k+1}^{i_1,(j)} = u_{k+1}^{i_2,(j)} = \dots = u_{k+1}^{i_\tau,(j)}$$

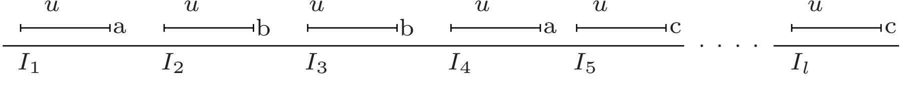


**Figure 4.6** Extension of $\hat{u}$ for local covers.

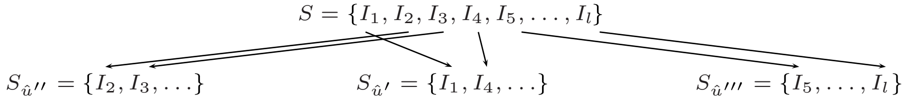

$S = \{I_1, I_2, I_3, I_4, I_5, \dots, I_l\}$
$S_{\hat{u}''} = \{I_2, I_3, \dots\}$
$S_{\hat{u}'} = \{I_1, I_4, \dots\}$
$S_{\hat{u}'''} = \{I_5, \dots, I_l\}$

**Figure 4.7** Partitioning the set $S$.

and let their starting positions

$$I = \{i_1, i_2 \dots i_\tau\}$$

Then, $\hat{u}_k^{(j)} = \{a | a \in u_{k+1}^{i,(j)}, i \in I\}$ and $\hat{u}_j = \hat{u}_1^{(j)} \hat{u}_2^{(j)} \dots \hat{u}_k^{(j)}$.

Step 4.5

Suppose in one position $I_\mu^{(j)}$, $u^{\mu,(j)}$ cannot be extended further without giving a mismatch, but matching substrings follow this unmatched substring. Therefore, we want to discard this $u_\mu$ and cover the string with the rest of the substrings $u_i$. We do this by first deleting $I_\mu^{(j)}$ from $S_1^{(j)}$. Then we delete and update the distances between $g_{\mu-1}^{(j)}$ and $g_{\mu+1}^{(j)}$ in $S_2^{(j)}$.

$$S_1^{(j)} = \{I_1^{(j)}, I_2^{(j)}, \dots I_{\mu-1}^{(j)}, \cancel{I_\mu^{(j)}}, I_{\mu+1}^{(j)} \dots I_\ell^{(j)}\}$$
$$S_2^{(j)} = \{g_1^{(j)}, g_2^{(j)}, \dots g_{\mu-1}^{(j)}, \cancel{g_\mu^{(j)}}, g_{\mu+1}^{(j)} \dots g_\ell^{(j)}\}$$

By keeping the linked list $L$ and from Theorem 4.1, the corresponding distances $g_{\mu-1}^{(j)}$ and $g_{\mu+1}^{(j)}$ in $S_2^{(j)}$ can be found in constant time.

The nonextendable occurrences, say $u^\gamma$ get a new set of data structures $S_1^{(\gamma)}$, $S_2^{(\gamma)}$, $S_3^{(\gamma)}$, and LC.

Step 4.6

We now have to update the LC array. After the extension of $\hat{u}^{(j)}$, two cases are possible:

*Case 4.1* An occurrence of $\hat{u}^{(j)}$ within a local cover cannot be extended by the same character as all other occurrences of $\hat{u}^{(j)}$ in the same local cover. In this case,

1. The local cover of length $k$ is maximal
2. The local cover of length $k + 1$ will be split into two smaller local covers and $\Lambda$ updated accordingly (see Figure 4.8).

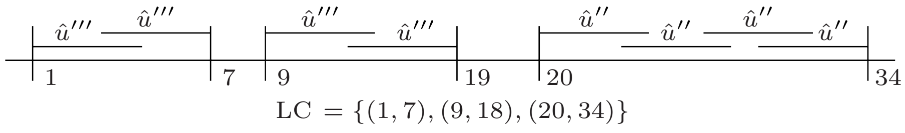


**Figure 4.8** LC after removal of third occurrence of $\hat{u}'$.

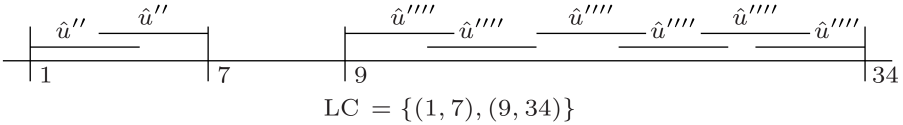


**Figure 4.9** LC after extension of $u'''$.

*Case 4.2* An occurrence of $\hat{u}^{(j)}$ can be extended beyond the end position of the local cover to which it belongs. In this case, if the condition $g_{\hat{u}}^{\prime(j)} \leq k+1$ is met, then the two local covers are joined together to make a larger local cover, and $\Lambda$ is updated accordingly (see Figure 4.9).

### 4.4.4 Computing All Covers of *x*

To find all covers of the string $x$, we slightly modify Algorithm 4.4.2. Instead of stopping when we have $g_\ell$, we continue increasing the length of $u$, until $|u| = n-1$, where $n = |x|$. During every iteration of the Main Step, if $g_\ell < |u|$, then we output $u$, as it is a cover of $x$.

### 4.4.5 Computing the Seeds of *x*

A substring $w$ of $x$ is called a seed of $x$ if $w$ covers one superstring of $x$ (this can be any superstring of $x$, including $x$ itself). For example, *aba* and *ababa* are some seeds of $x = ababaab$.

If a substring $u^i$ is a seed of a string $x$, then a superstring $y$, exists $y = sxv$, $|s| < |u|$ and $|v| < |u|$, which can be constructed by overlapping or concatenating copies of the strings $u^1, u^2, u^3, \ldots, u^\ell$.

By the definition of seeds, $x[i \ldots n]$ can be matched to any prefix of $u^i$, and $x[1 \ldots j]$ can be matched to any suffix of $u^i$.

With the previous algorithm, we find the covers $\hat{u}$ of string $x$. Therefore, we extend the previous algorithm by one more test, which tests whether $\hat{u}$ is also a seed of $x$.

We want to check whether cover $\hat{u}$ is also a seed of $x$. We would do this by checking whether $x[i \ldots n]$ can be matched to any prefix of $\hat{u}$, and whether $x[1 \ldots j]$ can be matched to any suffix of $\hat{u}$. In position $I_1$, where we have the first occurrence of $\hat{u}$, $u^1$, we test whether $u^1$ is a suffix of $x$. Additionally, we test whether the last occurrence of $\hat{u}$, $u^\ell$, is a prefix of $x$. If these two sentences are true, then the cover $\hat{u}$ with the sequence of $u^1, u^2 \ldots u^\ell$ of substrings is also a seed of $x$, as it can form a superstring of $x$, as shown in Figure 4.10.

## 4.5 CONSERVATIVE STRING COVERING IN DEGENERATE STRINGS

In this section, we describe algorithms for finding string regularities in constrained degenerate strings. Section 4.5.1 describes the algorithm for conservative pattern matching. Additionally, Section 4.5.2 and Section 4.5.3 describe the algorithms for computing conservative covers and seeds of a string, respectively.

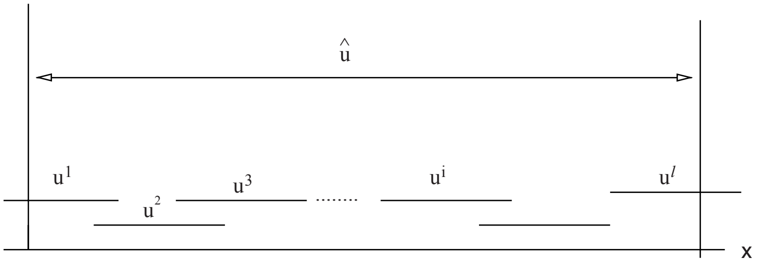


**Figure 4.10** Finding seeds of string *x*.

### 4.5.1 Finding Constrained Pattern *p* in Degenerate String *T*

As a building step, here, we study the constrained pattern matching problem on degenerate strings. The problem of constrained degenerate pattern matching is defined as follows:

Input: We are given a pattern $p$ of length $m$ with at most $\kappa$ nonsolid symbols, where $\kappa$ is a constant. We are given a degenerate string $T$, the text of length $n$.
Query: Find all occurrences of the pattern $p$ in the text $T$, (*i.e.*, find the positions in $T$ where the intersection of the pattern and the text is nonempty).

■ **EXAMPLE 4.1**

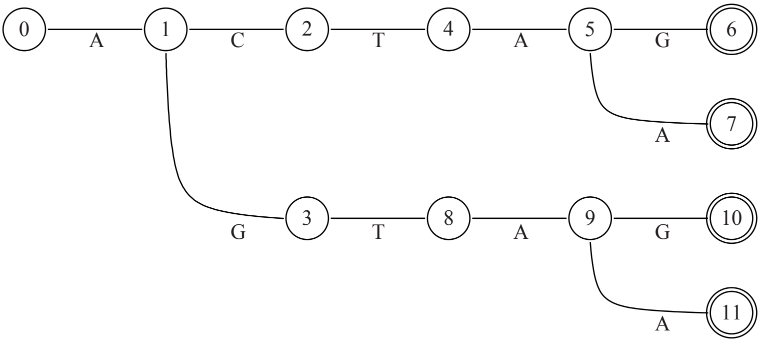
We consider a pattern, $p =$ A[CG]TA[AG] and text, $T =$ GA[CG][CT]AG[AT]A [AG][CT][AT]AG. Figure 4.11 shows the result of searching for $p$ in $t$. It is shown in the figure that $p$ occurs in $t$ starting at positions 2, 5, 8, and 9.

The algorithm works in two steps:

Step 4.7
Let the pattern $p$ be $p = P_1P_2 \dots P_m$. We built the Aho-Corasick automaton for the dictionary of the prefixes of the pattern $D = \{\pi_1\pi_2 \dots \pi_m, \forall \pi_i \in P_i, 1 \leq i \leq m\}$. Note that $|D| = \prod_{i=1}^{m} |P_i| < 2^{\kappa}$, as there are at most $\kappa$ nonsolid symbols.

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| t | G | A | [CG] | [CT] | A | G | [AT] | A | [AG] | [CT] | [AT] | A | G |
| Matches | | A | [CG] | T | A | [AG] | | | | | | | |
| | | | | | A | [CG] | T | A | [AG] | | | | |
| | | | | | | | | A | [CG] | T | A | [AG] | |
| | | | | | | | | | A | [CG] | T | A | [AG] |

**Figure 4.11** Pattern matching with *p* and *t*.



| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f(i)$ | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 1 | 0 | 1 | 3 | 1 |

**Figure 4.12** Aho-Corasick automata and its failure function for *p*.

Step 4.8
Assume that we have processed $T[1, i]$. At this point, we have a set, $P$, of prefixes of the strings in the dictionary in the Aho-Corasick automaton. We now will perform iteration $i + 1$. For each symbol $\tau$ occurring at $T[i + 1]$, we try to extend each prefix in $P$ by that symbol $\tau$, or we follow its failure link provided by the Aho-Corasick automaton. Figures 4.12 and 4.13 present a part of the matching process for the previous example.

Note that $|P|$ is bounded by the maximum number of possible prefixes, which in turn, is bounded by the size of the automaton. Thus, this method is linear.

### 4.5.2 Computing λ-Conservative Covers of Degenerate Strings

Here, we study another string regularity—a conservative covering of a degenerate string with a fixed length cover. The λ-conservative cover problem is defined as follows:

Input: We are given a conservative degenerate string $t$, of length $n$, a constant $\kappa$, which is the maximum number of nonsolid symbols allowed in a cover, and an integer $\lambda$, which is the length of the cover.
Query: Is there a conservative cover, $c$, of $t$ of length $\lambda$?

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|---|
| $t$ | G | A | [CG] | [CT] | A | G | [AT] | … |
| $P$ | 0 | {1} | {2,3} | {4,8} | {5,9} | {6, 10} | {8} | … |

**Figure 4.13** Matches of prefixes of *P* in text *t*.

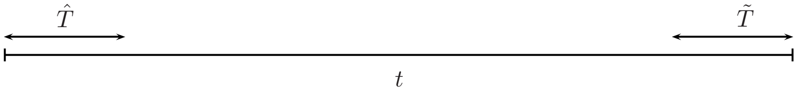


**Figure 4.14** The cover, *c*, covers the beginning and the end of *T*. Thus, $\hat{T}$ and $\tilde{T}$ provide the set of potential candidates.

STEP 4.9
We consider the prefix, $\hat{T}$, of $t$ of length $\lambda$ (see Figure 4.14),

$$\hat{T} = T_1 \dots T_\lambda$$

and the suffix, $\tilde{T}$ of $t$ of length $\lambda$,

$$\tilde{T} = T_{n-\lambda+1}, \dots T_n$$

We build the Aho-Corasick automaton for the dictionary

$$D = \{t_1 \dots t_\lambda \mid \forall\, t_i \in T_i \cap T_{i+n-\lambda},\ 1 \le i \le \lambda\}$$

STEP 4.10
For each $d \in D$, we find all of its occurrences in $T$, parsing the text $T$ through the Aho-Corasick Automaton built in STEP 4.9. If a word $d$ occurs at position $i$, then we set a flag $L(i) =$ true. If the distance $|i - j|$ of any two consecutive flags is less than $\lambda$, then we have a cover

$$C_1 C_2 \dots C_\lambda, \text{ where}$$

$$C_i = \{d_i, \text{ is the } i\text{-th letter of every word in } D,\ 1 \le i \le \lambda\}$$

The overall complexity of these two steps is linear.

### 4.5.3 Computing λ-Conservative Seeds of Degenerate Strings

Here we study yet another regularity, covering a degenerate string with a seed of a given length. The $\lambda$-constrained seed problem is defined as follows:

INPUT: We are given a degenerate string $t$, of length $n$, a constant $\kappa$, which is the maximum number of nonsolid symbols allowed in a seed and an integer $\lambda$, which is the length of the seed. Figure 4.15 presents an example of a conservative seed, $\hat{s}$, a seed of the string $t$.
QUERY: Is there a conservative seed, $s$, of $t$ of length $\lambda$?

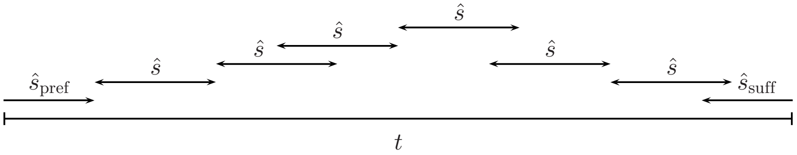


**Figure 4.15** Above, $\hat{s}$ is a seed of the string $t$, where each $\hat{s}$ contains at most $\kappa$ nonsolid symbols and is of length $\lambda$. Also, $\hat{s}_{\text{pref}}$ and $\hat{s}_{\text{suff}}$ are a prefix and suffix of $\hat{s}$, respectively.

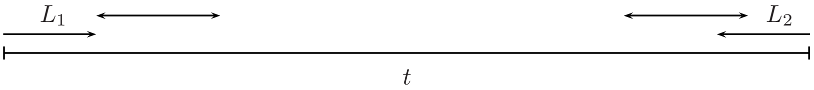


**Figure 4.16** The positions of candidate seeds from lists $L_1$ and $L_2$.

STEP 4.11
The first occurrence of the seed can be in any of the positions $\{1 \dots \lambda\}$. Figure 4.16 presents positions of candidate seeds. Thus, we consider the following strings of length $\lambda$:

$$L_1 = \{T[1..\lambda], T[2..\lambda + 1], \dots, T[\lambda..2\lambda - 1]\}$$

and all suffixes of string $t$ of length $\lambda$:

$$L_2 = \{T[n - \lambda..n], T[n - \lambda - 1..n - 1], \dots, T[n - 2\lambda - 1]\}$$

We build the Aho-Corasick automaton for the dictionary

$$D = \{t_{i_1} \dots t_{i_\lambda} | \forall t_{i_j}, \text{ where } t_{i_j} \text{ is the } j\text{-th symbol of } T \in \mathrm{L}_1 \cup \mathrm{L}_2\}.$$

STEP 4.12
For each $d \in D$, we find all of its occurrences in $T$, parsing the text $T$ through the Aho-Corasick Automaton built in STEP 4.11. If a word $d$ occurs at position $i$, then we set a flag $L_d(i) =$ true. If the distance $|i - j|$ of any two consecutive flags in $L_d$ is less than $\lambda$, then $d$ is a candidate for a seed. Let $i_1$ and $i_2$ be the first and last occurrences of $d$ in $T$. We check whether $T[1, i_1]$ is a suffix of $d$ and whether $T[i_2, n]$ is a prefix of $d$; if that is the case, then $d$ is a seed. The overall complexity of the algorithm is $O(\lambda n)$.

## 4.6 CONCLUSION

In this chapter, we have seen recent advances in algorithms for degenerate strings that can be applied for computational biology problems. We have shown $O(n \log n)$ algorithms for finding the smallest cover, local covers, and all covers of a string. We also have presented a $O(n \log n)$ algorithm for finding the seeds of a string. Additionally, we have shown $O(n)$ algorithms for finding the smallest conservative cover, $\lambda$-conservative local covers. We also have presented a $O(\lambda n)$ algorithm for finding the $\lambda$-conservative seeds of a string. All algorithms we have used are easily adaptable to allow the bit-matching techniques to be used to allow efficient implementations.

## REFERENCES

1. A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun ACM*, 18(6):333–340, 1975.
2. A. Apostolico and D. Breslauer. An optimal $O(log\,logn)$-time parallel algorithm for detecting all squares in a string. *SIAM J Comput*, 25(6):1318–1331, 1996.
3. A. Apostolico, M. Farach, and C.S. Iliopoulos. Optimal superprimitivity testing for strings. *Inf Process Lett*, 39:17–20, 1991.
4. R. Baeza-Yates and G. Gonnet. A new approach to text searching. *Commun ACM*, 35:74–82, 1992.
5. O. Berkman, C.S. Iliopoulos, and K. Park. The subtree max gap problem with application to parallel string covering. *Inf Comput*, 123(1):127–137, 1995.
6. M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf Process Lett*, 12(5):244–250, 1981.
7. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, Cambridge, U.K., 2007.
8. M.J. Fischer and M.S. Paterson. String-matching and other products. Technical report, Cambridge, MA, 1974.
9. J. Holub, W.F. Smyth, and S. Wang. Fast pattern-matching on indeterminate strings. *J Discrete Algorithm*, 6(1):37–50, 2008.
10. J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison Wesley, Reading, MA, 2000.
11. C.S. Iliopoulos, D.W.G. Moore, and K. Park. Covering a string. *Proceedings of the 4-th Symposium on Combinatorial Pattern Matching*, volume 684 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1993, pp. 54–62.
12. H. Imai and T. Asano. Dynamic orthogonal segment intersection search. *J Algorithm*, 8(1):1–18, 1987.
13. D.E. Knuth, J. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J Comput*, 6(2):323–350, 1977.
14. C. Linhart and R. Shamir. The degenerate primer design problem: Theory and applications. *J Comput Bio*, 12(4):431–456, 2005.
15. D. Moore and W.F. Smyth. An optimal algorithm to compute all the covers of a string. *Inf Process Lett*, 50(5):239–246, 1994.
16. M.C. Shaner, I.M. Blair, and T.D. Schneider. Sequence logos: A powerful, yet simple, tool. In T.N. Mudge, V. Milutinovic, and L. Hunter, editors, *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences, Volume 1: Architecture and Biotechnology Computing*, IEEE Computer Society Press, New York, NJ, 1993, pp. 813–821.

17. M.V.T. Flouri and L. Vagner. Indexing degenerate strings. *Proceedings of International Conference of Computational Methods in Sciences and Engineering*, American Institute of Physics, Melville, New York, 2007, pp. 1400–1403.

18. S. Wu and U. Manber. Agrep – a fast approximate pattern-matching tool. *Proceedings USENIX Winter 1992 Technical Conference*, San Francisco, CA, 1992, pp. 153–162.

19. S. Wu and U. Manber. Fast text searching: Allowing errors. *Commun ACM*, 35(10):83–91, 1992.

# 5

# EXACT SEARCH ALGORITHMS FOR BIOLOGICAL SEQUENCES

Eric Rivals, Leena Salmela, and Jorma Tarhio

## 5.1 INTRODUCTION

With the development of sequencing techniques, it has become easy to obtain the sequence (*i.e.*, the linear arrangement of residues [nucleotides or amino-acids]), of DNA, RNA, or protein molecules. However, determining the function of a molecule remains difficult and is often bound to finding a sequence similarity to another molecule whose role in the cell is at least partially known. Then the biologist can predict that both molecules share the same function and try to check this experimentally. *Functional annotations* are transferred from one sequence to another provided that their similarity is high enough. This procedure is also applied to molecule subparts, whose sequences are shorter; such as protein domains, DNA/RNA motifs, and so on.

Depending on the sequence lengths and the expected level of evolutionary relatedness, the sequence similarity can be found using alignment or pattern matching procedures. A quest in bioinformatics has been to design more sensitive sequence similarity searching methods to push further the limit or *gray zone* at which evolutionary sequence similarity cannot be departed from random sequence similarity [4,21]. These methods (*e.g.*, profile hidden Markov Models) have provided, at the expense of computing time, important improvements in functional annotations. However, it has soon become clear that in other frameworks, only high-level similarity

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

was sought, and speed rather than sensitivity was the major issue. Hence, researchers have designed the following continuum of methods that can be classified according to the level of allowed dissimilarity:

1. Full sensitivity alignment (Smith and Waterman algorithm [58]),
2. Fast similarity search programs (*e.g.*, BLAST [4]),
3. Approximate pattern matching (*e.g.*, BOWTIE [36]),
4. Near-exact and exact pattern matching (*e.g.*, MPSCAN [53]).

For some everyday sequence manipulation tasks, the user needs exact pattern matching programs (as available in large bioinformatic program suites like EMBOSS) to find from which chromosome or where in a genome a given sequence comes from; to find short nucleotidic motifs, like restriction or cleavage sites, in long DNA sequences; to verify whether a *distinguishing* sequence motif really separates negative from positive instances (longer sequences). The latter happens when designing oligonucleotides for gene expression arrays or multiple primers for multiplex polymerase chain reactions [50]. Even for exploring protein sequences, a server has been launched that offers an exact search for short polymers in all sequences of protein databanks [9]. In such frameworks, the need is for a single or multiple pattern search for a few hundreds patterns, which can be solved easily by repetitively applying a single pattern matching program. Algorithmic solutions for these tasks will be described in Section 5.2. However, pattern matching algorithms fail to become popular among biologists for several reasons as follows:

- Most of them lack implementations capable of handling biological sequence formats (which then requires to change the format).
- They lack a graphical interface or were not integrated in popular graphical sequence exploration package like the Genetics Computer Group (GCG) package.
- As BLAST [4] was used for similarity searching on a daily basis, it has become the all-purpose tool for most sequence processing tasks, even when more adapted solutions were available [15].

Since 2005, biology has experienced the revolution of *high-throughput sequencing* (HTS) because of the renewal of sequencing techniques (new technologies often are termed *next generation sequencing*) [43]. Because of the invention of parallel sequencing of multiple molecules on a single machine, the sequencing output per run has grown by orders of magnitude compared with the traditional Sanger technique and is expected to increase further [20]. This change does not only have technological consequences. Experiments previously done by hybridization now are performed preferentially by sequencing [10], because these techniques offer a much deeper sampling and allow covering the whole genome. Hence, HTS now is exploited to address surprisingly diverse biological questions of genome sequencing or resequencing [5,43], transcriptomics [49,59], genomic variation identification or genome breakpoint mapping [14], metagenomics [24], and epigenomics [12, 26]. To grasp how drastic the shift is, consider that one epigenomic census assay published in 2007 produced in one experiment an already amazing 1.5 million short read sequences of 27 base pairs (bp)[1] each [27], whereas another assay published only one year later delivered with the same technology 15 million sequences of 20 bp reads [12].

In all such applications, the first bioinformatic task is to *map* the short reads on a reference genome sequence or on a large collection of DNA sequences. The goal of mapping in transcriptomics, epigenomics, and other applications is to point out chromosomic positions either transcribed [59], bound to a protein [27], or whose three-dimensional conformation is altered by a protein [12]. Hence, further analysis only considers those reads that mapped to a unique genomic positions. In other frameworks, all mapped reads inclusive of those mapped at multiple positions provide important information to detect, for example, new copies of repeats in the sampled genome. The number of (uniquely and/or multi-) mapped reads depends on the read length, on the expected probability for a read to map on the genome, on the level of sequence errors in the reads, as well as on the genetic differences between the cell from which the reads were sequenced and that which provided the reference genome sequence. The following approaches are possible: to map exactly or approximately (up to a limited number of differences between the read and the genome sequences) reads on the genome sequence. The choice between the two is not obvious because it has been shown for instance that exact mapping with a shorter read length can yield the same number of uniquely mapped reads than approximate matching allowing up to two mismatches [53] and because all approximate mapping tools are not based on the same algorithm [26,36,39,40,57]. If approximate mapping is used, then another question is how to distinguish a difference resulting from genetic variation or from sequence error in a match?

More practically, whether sequence quality information is provided aside the reads themselves, often the complete read sequence cannot be exploited because of low quality positions. Hence, either preprocessing with various parameters is applied to eliminate some positions, or multiple mappings with different parameters are tested to optimize the mapping output. In any case, the number of reads to map is so large that mapping efficiency and scalability, both in terms of time and to a less extent of memory, becomes a major issue. In Section 5.4, we will discuss the comparison of exact versus approximate mapping approaches on these issues. Before that, Section 5.2 presents efficient solutions for the *single pattern matching problem*, whereas Section 5.3 details fast algorithms for *multiple* or *set pattern matching*.

## 5.2 SINGLE PATTERN MATCHING ALGORITHMS

We consider locating nucleotide or amino acid sequence patterns in a long biological sequence called the text. We assume that the sequences are in the raw format. We denote the pattern of length $m$ by $P = p_0 p_1 \dots p_{m-1}$ and the text of length $n$ by

[1] A base pair is the length unit of a DNA/RNA sequence.

$T = t_0 t_1 \ldots t_{n-1}$. We also use C-like notations |, &, and, $\ll$ to represent bitwise operations OR, AND, and left shift, respectively. The goal of the *single pattern matching problem* is to find all *occurrences* of the pattern in the text (*i.e.*, positions $j$ such that $t_{j+i} = p_i$ for $i = 0, 1, \ldots, m - 1$).

### 5.2.1 Algorithms for DNA Sequences

Most efficient string matching algorithms in the DNA alphabet are modifications of the Boyer–Moore algorithm [11], which processes the text in windows of length $m$. In each window, the characters are read from right to left, and when a mismatch with the pattern is found, the window is shifted based on the text characters read. The algorithm applies two shifting heuristics, match and occurrence. The *match heuristic* assures that the matching suffix of the current window matches the pattern also after the shift if it then is aligned with the pattern. The *occurrence heuristic* (also called the bad character heuristic) determines the shortest possible shift such that either the mismatching or the right-most character of the current window matches the pattern after the shift. If no such shift is possible, then a shift of length $m$ is taken. In most modifications of the Boyer–Moore algorithm, only the occurrence heuristic is applied for shifting. The Boyer–Moore–Horspool algorithm [22] (BMH) is a famous implementation of this simplification.

Because the DNA alphabet contains only four symbols, shifts based on one character are short on average. Therefore, it is advantageous to apply $q$-mers (or $q$-grams), strings of $q$ characters, for shifting instead of single characters. This technique was mentioned already in the original paper of Boyer and Moore [11, p. 772], and Knuth *et al.* [34, p. 341] theoretically analyzed its gain. Zhu and Takaoka [65] presented the first algorithm using the idea. Their algorithm uses two characters for indexing a two-dimensional array. Later, Baeza-Yates [6] introduced another variation based on the BMH algorithm in which the shift array is indexed with an integer formed from a $q$-mer with shift and add instructions.

For the DNA alphabet, Kim and Shawe-Taylor [32] introduced a convenient alphabet compression by masking the three lowest bits of ASCII characters. In addition to the a, c, g, and t, one gets distinguishable codes also for n and u. Even the important control code \n = LF has a distinct value, but \r = CR gets the same code as u. With this method, they could use $q$-mers of up to six characters. Indexing of the shift array is similar to that of Baeza-Yates' algorithm.

With the DNA alphabet, the probability of an arbitrary short $q$-mer appearing in a long pattern is high. This restricts the average shift length. Kim and Shawe-Taylor [32] introduced a variation for the cases in which the $q$-mer in the text occurs in the pattern. Then two additional characters are checked one by one to achieve a longer shift.

In most cases, the $q$-mer that is taken from the text does not match with the last $q$-mer of the pattern, and the pattern can be shifted forward. For efficiency, one can apply a *skip loop* [23] in which the pattern is moved forward until the last $q$-mer of the pattern matches with a $q$-mer in the text. The easiest way to implement this idea is to place a copy of the pattern as a stopper after the text and artificially define the shift of the last $q$-mer of the pattern to be zero. Then the skip loop is exited when the shift is zero. After a skip loop, the rest of the pattern is compared with the corresponding text positions.

A crucial point for the efficiency of a $q$-mer algorithm is how $q$-mers are computed. Tarhio and Peltola [61] presented a $q$-mer variation of BMH that applies a skip loop. The algorithm computes an integer called a fingerprint from a $q$-mer. The ASCII codes are mapped to the range of 4: $0 \leq r[x] \leq 3$, where $r[x]$ is the new code of $x$, such that characters a, c, g, and t get different codes, and other possible characters get, for example, code 0. In this way, the computation is limited to the effective alphabet of four characters. The fingerprint is simply a reversed number of base. A separate transformation table $h_i$ is used for each position $i$ of a $q$-mer, and multiplications are incorporated during preprocessing into the following tables: $h_i[x] = r[x] \times 4^i$. For $q = 4$, the fingerprint of $x_0 \cdots x_3$ is $\sum_{i=0}^{3} r[x_i] \times 4^i$, which then is computed as

$$h_0[x_0] + h_1[x_1] + h_2[x_2] + h_3[x_3]$$

Recently, Lecroq [37] presented a related algorithm. Its implementation is based on the Wu–Manber algorithm [63] for multiple string matching, but as suggested, the idea is older [11,65]. For $q = 4$, the fingerprint of $x_0 \cdots x_3$ is

$$((((((x_0 \ll 1) + x_1) \ll 1) + x_2) \ll 1) + x_3) \bmod 256$$

SSABS [56] and TVSBS [62] were developed with biological sequences in mind. SSABS is a Boyer–Moore-type algorithm. In the search phase, the algorithm verifies that the first and last characters of the pattern match with the current alignment before checking the rest of the alignment (or guard tests). TVSBS uses a 2-mer for calculating the shift, adopted from the Berry–Ravindran algorithm [8], which is a cross of the Zhu–Takaoka algorithm and Sunday's Quick Search algorithm [60]. Instead of the two-dimensional shift table of Berry–Ravindran, TVSBS uses a hash function to compute an index to a one-dimensional table. According to Kalsi *et al.* [28], SSABS and TVSBS are not competitive with $q$-mer algorithms in the DNA alphabet.

We present one fast $q$-mer algorithm for DNA sequences in detail. It is SBNDM4 [19], a tuned version of backward nondeterministic DAWG matching (BNDM) by Navarro and Raffinot [45]. BNDM is a kind of cross of the backward DAWG matching algorithm (BDM) [16] and the Shift-Or [7] algorithm. The idea of BNDM is similar to BDM, although instead of building a deterministic automaton, a nondeterministic automaton is simulated even without constructing it. The resulting code applies bit-parallelism, and it is both efficient and compact. We present a pseudo code[2] for SBNDM4 as Algorithm 5.1. The code contains a skip loop so that a copy of the pattern is placed to $t_n \ldots t_{n+m-1}$ on line 3. The presented version outputs only the number of matches. The matches can be reported by changing line 13.

[2] The conditional expressions of the while statements on lines 6 and 9 contain side assignments. Thus, the operators = of these expressions are not relational operators but assignment operators.

**Algorithm 5.1**

```
SBNDM4($P = p_0p_1 \dots p_{m-1}, T = t_0t_1 \dots t_{n-1}$)
1.  for ($i = 0$; $i < 256$; $i = i + 1$) $B[i] = 0$
2.  for ($i = 0$; $i < m$; $i = i + 1$) $B[p_{m-i-1}] \mid= (1 \ll i)$
3.  for ($i = 0$; $i < m$; $i = i + 1$) $t_{n+i} = p_i$
4.  $j = m - 1$
5.  while true
6.      while not ($d = ((B[t_j] \ll 3)\ \&\ (B[t_{j-1}] \ll 2)\ \&\ (B[t_{j-2}] \ll 1)\ \&\ B[t_{j-3}])$)
7.          $j = j + m - 3$
8.      $pos = j$
9.      while ($d = ((d \ll 1)\ \&\ B[t_{j-4}])$) $j = j - 1$
10.     $j = j + m - 4$
11.     if $j = pos$
12.         if ($j \geq n$) return nmatch
13.         nmatch = nmatch + 1
14.         $j = j + 1$
```

SBNDM4 is very fast in practice. On x86 processors, one still can boost its performance by using 16-bit reading [19]. In searching DNA patterns of 20 characters, SBNDM4 with 16-bit reading is more than eight times faster than the classical Boyer–Moore algorithm [19] (see also comparisons [28, 37, 61]). SBNDM4 works for DNA patterns of up to 32 or 64 characters depending on the word size of the processor. In practice, longer exact patterns are seldom interesting, but, for example, Lecroq's algorithm [37], with the divisor 4096 instead of 256, is good for them. SBNDM4, like other variations of BNDM, also works for more general string matching in which positions in the pattern or in the text represent character classes [46] instead of single characters. So, for example, the standard IUB/IUPAC nucleic acid codes [66] can be used with SBNDM4.

There are also algorithms [33, 51] for packed DNA. We decided to leave them outside this presentation.

### 5.2.2 Algorithms for Amino Acids

In general, there is hardly any difference in performance when searching amino acid or natural language patterns. So any good search algorithm for natural language is also applicable to amino acids. In searching short patterns [19], SBNDM2, the 2-mer variation of SBNMD4, is among the best. SBNDM2 is derived from SBNDM4 as follows: replace line 6 with

$$\textbf{while}\ \texttt{not}\ (d = ((B[t_j] \ll 1)\ \&\ B[t_{j-1}]))$$

and on lines 7–10, replace $m - 3$ with $m - 1$, $j - 4$ with $j - 2$, and $m - 4$ with $m - 2$, respectively.

As for SBNDM4, one still can boost the performance of SBNDM2 by using 16-bit reading [19]. In searching patterns of five characters, SBNDM2 with 16-bit reading is more than two times faster than the classical Boyer–Moore algorithm.

## 5.3 ALGORITHMS FOR MULTIPLE PATTERNS

In this section, we consider *exact searching of multiple patterns*. More precisely, we are given a text and $r$ patterns, and we need to find all occurrences of all patterns in the text. Here, we focus on algorithms that can be run on standard hardware. Some attempts also have been made to implement standard set pattern matching algorithms on specific parallel hardware to gain computing time [18].

### 5.3.1 Trie-Based Algorithms

Many algorithms for exact searching of multiple patterns are based on a data structure called trie for storing the patterns. A trie is a tree in which each edge is labeled with a character. Each node of the trie is associated with a string that is formed by concatenating all the labels of the edges on the path from the root to the node. Given a node in the trie, all edges to the children of this node have a different label. Figure 5.1 shows the trie storing the strings {acc, accg, att, cca, cgt}.

***5.3.1.1 Aho–Corasick.*** The Aho–Corasick algorithm [2] builds as preprocessing an automaton that recognizes the occurrences of all patterns. The preprocessing starts by building the trie of the pattern set. We add an edge from the root to the root for all those characters that do not yet have an outgoing edge from the root. The trie then is augmented with *failure links* as follows. The failure link of a node $N$ in the trie points to a node that is associated with the longest possible suffix of the string

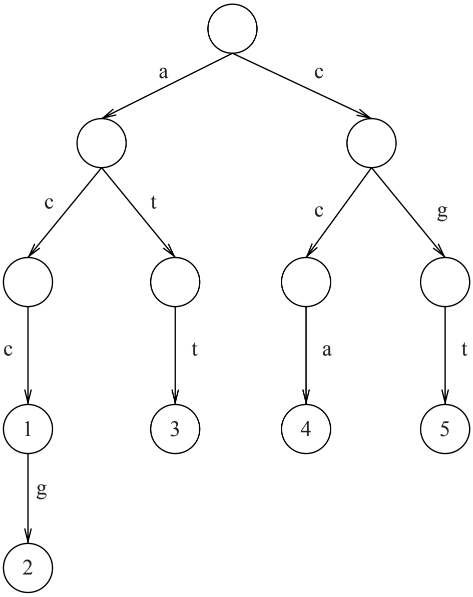


**Figure 5.1** An example trie storing the strings {acc, accg, att, cca, cgt}.

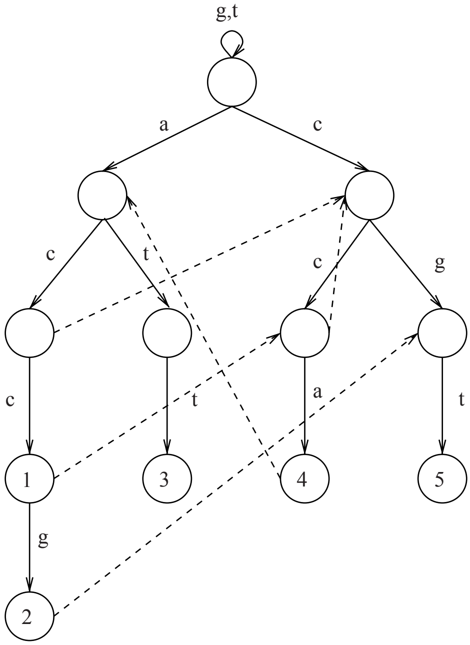


**Figure 5.2** An example of the Aho–Corasick automaton for the patterns {acc, accg, att, cca, cgt}. The dashed lines show the failure links. Failure links to the starting state of the automaton have been omitted. The numbers inside the nodes show the values of the output function.

associated with the node $N$ excluding the node $N$ itself. Additionally, we associate an *output function* with each node whose associated string is one of the patterns. The output function outputs the identifier of the pattern. Figure 5.2 shows an example of an Aho–Corasick automaton for the patterns {acc, accg, att, cca, cgt}.

The automaton is used for searching the text as follows. We start at the root node of the trie. We read the text character by character, and for each character, we perform the following actions. As long as there is no child node with an edge labeled with the read character, we follow the failure link. Then we descend to the child with an edge labeled with the read character. Finally, we output the identifiers returned by the output function for the child node.

The Aho–Corasick automaton can be built in $\mathcal{O}(cM)$ time, where $c$ is the size of the alphabet and $M$ is the total length of the pattern set (*i.e.*, $M = r \times m$ if all patterns are of length $m$). The searching phase takes $\mathcal{O}(n + \text{occ})$ time, where occ is the total number of occurrences of all patterns in the text.

***5.3.1.2 Set Backward Oracle Matching.*** The set backward oracle matching (SBOM) algorithm [3] builds an automaton that recognizes at least all *factors* (*i.e.*, substrings) of the reversed patterns. The automaton is built as shown in Algorithm. 5.2. First we build a trie of the reversed patterns. Then we traverse the trie in breath-first order and add some more edges between the nodes turning the trie into a

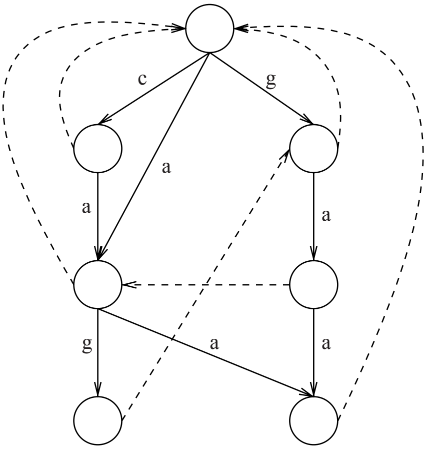


**Figure 5.3** An example of the SBOM automaton. The dashed lines show the supply links.

### Algorithm 5.2

```
SBOM-preprocess ($P_1, \ldots, P_r$)
1. build-trie($P_1^r, \ldots, P_r^r$)
2. set supply link of root to NULL
3. for each node $N$ in the trie in breath first order
4.   down $=$ supply link of parent
5.   $c$ $=$ the edge label from parent to $N$
6.   while down $\neq$ NULL and
        down does not have a child with edge label $c$
7.     add an edge from down to $N$ with label $c$
8.     down $=$ supply link of down
9.   if down $\neq$ NULL
10.    set supply link of $N$ to the child of down with edge label $c$
11.  else
12.    set supply link of $N$ to root
```

*directed acyclic graph* (DAG) that recognizes all factors of the reversed patterns. To assist us in adding these new edges, we associate a *supply link* with each node. For each node we then perform the pseudo code on lines 4–12 shown in Algorithm 5.2. Figure 5.3 shows an example of the SBOM automaton built for patterns {aag, gac}. As is shown, the automaton also recognizes some other strings, like caa, which are not factors of the patterns.

This automaton then is used for searching the occurrences of the patterns as follows. Initially, we set the endpoint to the length of the shortest pattern. We then read the characters of the text backward, starting at the endpoint character. For each character, we make the corresponding state transition in the automaton. Whenever we encounter a node associated with one of the patterns, we verify the read region character by character against the pattern. If there is no transition from the current state with the read character, we can shift the endpoint forward and start the backward scan again at the new endpoint. The length of the shift is 1 or $m - j + 1$, where $m$ is the length of the shortest pattern and $j$ is the number of characters that we have read—whichever is longer.

The SBOM automaton can be built in $\mathcal{O}(M)$ time, where $M$ is the total length of the pattern set. The worst-case searching time in SBOM is $\mathcal{O}(Mn)$, but on average, SBOM does not inspect every character of the text.

### 5.3.2 Filtering Algorithms

*Filtration* aims at eliminating most positions that cannot match any given pattern with an easy criterion. Then, *verification* checks whether the remaining positions truly match a pattern. Thus, filtering algorithms operate in three phases. The patterns first are preprocessed; in the second phase, we search the text with a filtering method, and the candidate matches produced by the filtering are verified in the third phase.

Here, we describe several algorithms that use a *generalized pattern* of *character classes* for filtration [55]. Let us explain the filtration scheme with the following example in which we have a set of three patterns of length $m = 8$: $\{P_1, P_2, P_3\} =$ {accttggc, gtcttggc, accttcca}, and we set $q$ to 5. The overlapping 5-mers (or 5-grams) of each pattern are given in Figure 5.4. For a text window $W$ of length 8 to match $P_1$, the substring of length $q$ starting at position $i$ in $W$ must match the $i$-th $q$-mer of $P_1$ for all possible $i$ and conversely. Now, we want to filter out windows that do not match any pattern. If the substring starting at position $i$ in $W$ does not match the $i$-th $q$-mer of neither $P_1$, $P_2$, nor $P_3$, then we are sure that $W$ cannot match any patterns. Thus, our filtration criterion to eliminate surely any nonmatching window $W$ is to find whether a position $i$ exists such that the previous condition is true.

$$\{P_1, P_2, P_3\} = \{\text{accttggc; gtcttggc; accttcca}\}$$

(a)

```
    1 2 3 4 5 6 7 8        1 2 3 4 5 6 7 8        1 2 3 4 5 6 7 8
P1  a c c t t g g c    P2  g t c t t g g c    P3  a c c t t c c a
    a c c t t              g t c t t              a c c t t
      c c t t g              t c t t g              c c t t c
        c t t g g              c t t g g              c t t c c
          t t g g c              t t g g c              t t c c a
```

(b)

$$[\text{acctt, gtctt}][\text{ccttg, tcttg, ccttc}][\text{cttgg, cttcc}][\text{ttggc, ttcca}]$$

(c)

**Figure 5.4** (a) A set of three patterns of length $m = 8$. (b) The overlapping 5-mers starting at position 1 to 4 (in very light gray, light gray, gray, dark gray, respectively) of each pattern. (c) The generalized 5-mer pattern for the set of patterns.

Given a set of patterns, the filtering algorithms build a single $q$-mer generalized pattern (Figure 5.4c). A generalized pattern allows several symbols to match at a position (like a position [DENQ] in a PROSITE pattern, which matches the symbols D, E, N, and Q). However, here each $q$-mer is processed as a single symbol. Then, a string matching algorithm that can handle classes of characters is used for searching for occurrences of the generalized pattern in the text.

Various different algorithms can be used for implementing the filtering phase. Subsequently we describe in more detail algorithms in which filtering is based on the shift-or, BNDM, and Boyer–Moore–Horspool algorithms. A filtering algorithm always requires an exact algorithm to verify the candidate matches. In principle, any presented exact algorithm could be used for this purpose.

We recently have shown that the average time complexity of the filtering algorithm based on the BNDM or Boyer–Moore–Horspool algorithm for searching $r$ patterns of length $m$ in a text of length $n$ over an alphabet of size $c$ is $\mathcal{O}(n \log_c(rm)/m)$, provided that $q = \Theta(\log_c(rm))$ [53, 54]. As it was proved that the minimum time required is $\Omega(n \log_c(rm)/m)$ [44], these algorithms are asymptotically optimal on average.

#### 5.3.2.1 Multipattern Shift-Or with q-Grams.

The shift-or algorithm is extended easily to handle classes of characters in the pattern [1, 7], and thus, developing a filtering algorithm for multiple pattern matching is straightforward. The preprocessing phase now initializes the bit vectors for each $q$-mer as follows. The $i$-th bit is set to 0 if the given $q$-mer is included in the character class in the $i$-th position. Otherwise, the bit is set to 1. The filtering phase proceeds then exactly like the matching phase of the shift-or algorithm. Given this scheme, it is clear that all actual occurrences of the patterns in the text are candidates. However, there are also false positives, as the generalized pattern also matches other strings than the original patterns. We call this algorithm SOG (short for multipattern Shift-or with $q$-grams).

#### 5.3.2.2 Multipattern BNDM with q-Grams.

The second filtering algorithm is based on the BNDM algorithm by Navarro and Raffinot [45]. This algorithm has been extended to classes of characters in the same way as the shift-or algorithm. We call the resulting multiple pattern filtering algorithm BG (short for BNDM with $q$-grams). The bit vectors of the BNDM algorithm are initialized in the preprocessing phase so that the $i$-th bit is 1 if the corresponding $q$-mer is included in the character class of the reversed generalized pattern in position $i$. In the filtering phase, the matching is then done with these bit vectors. As with SOG, all match candidates reported by this algorithm must be verified.

#### 5.3.2.3 Multipattern Horspool with q-Grams.

The last of our algorithms uses a Boyer–Moore–Horspool [22] type method for matching the generalized pattern against the text. Strictly speaking, this algorithm does not handle character classes properly. It will return all those positions in which the generalized pattern matches and also some others. This algorithm is called HG (short for Horspool with $q$-grams).

5-mer tables:

| 1. | 2. | 3. | 4. |
|---|---|---|---|
| accct | accct | accct | accct |
| | ccctt | ccctt | ccctt |
| | | cctta | cctta |
| | | | cttaa |

**Figure 5.5** Data structures of the HG algorithm for the pattern “acccttaa”.

The preprocessing phase of HG constructs a bit table for each of the $m - q + 1$ positions in which a $q$-mer starts in the pattern. The first table keeps track of $q$-mers contained in the character class of the first position of the generalized pattern, the second table keeps track of $q$-mers contained in the character classes of the first and the second position in the generalized pattern, and so on. Finally, the $m - q + 1$-st table keeps track of characters contained in any character class of the generalized pattern. Figure 5.5 shows the four tables corresponding to the pattern “acccttaa” when using 5-mers.

These tables then can be used in the filtering phase as follows. First, the $m - q + 1$-st $q$-mer is compared with the $m - q + 1$-st table. If the $q$-mer does not appear in this table, then the $q$-mer cannot be contained in the character classes of positions $1 \ldots m - q + 1$ in the generalized pattern, and a shift of $m - q + 1$ characters can be made. If the character is found in this table, then the $m - q$-th character is compared with the $m - q$-th table. A shift of $m - q$ characters can be made if the character does not appear in this table and, therefore, not in any character class in the generalized pattern in positions $1, \ldots, m - q$. This process is continued until the algorithm has advanced to the first table and found a match candidate there. The pseudocode is shown as Algorithm 5.3. Given this procedure, it is clear that all positions matching the generalized pattern are found. However, other strings also will be reported as candidate matches.

## Algorithm 5.3

```
HG-matcher($T = t_0 \ldots t_{n-1}$, $n$)
$i = 0$
while $i \leq n - m$
  $j = m - q + 1$
  while true
    if not qMerTable[$j$][$t_{i+j-1 \ldots i+j+q-2}$]
      $i = i + j$
      break
    else if $j = 1$
      verify-match($i$)
      $i = i + 1$
      break
    else
      $j = j - 1$
```

### 5.3.3 Other Algorithms

Other algorithms for searching multiple patterns include the Commentz–Walter algorithm [17] with its variations, the Wu–Manber algorithm [63], and algorithms derived from the Rabin–Karp algorithm [29] for a single pattern.

## 5.4 APPLICATION OF EXACT SET PATTERN MATCHING FOR READ MAPPING AND COMPARISON WITH MAPPING TOOLS

Here, we concentrate on the question of set pattern matching and on its main current application—read mapping on genomic sequences. In most frameworks, millions of reads that originate from a genome have been sequenced using HTS. A read serves as a signature for a molecule or a chromosomic position. The goal of mapping is to find for each different read the chromosomic position of origin in the reference genome. As a read may be sequenced several times according to its number of occurrences in the biological sample, the number of different reads may be much smaller than the number of read sequences. For example, in a transcriptomic assay in which 2 million reads were sequenced, the read set contains $\simeq$ 440.000 elements [49]. As a read sequence can differ from the original chromosomic sequence because of polymorphisms or sequence errors, read mapping is often performed using approximate pattern matching, which allows a few mismatches and/or indels. For approximate mapping, either near-exact sequence similarity search programs (BLAT [30], MEGABLAST [64], or SSAHA [48]) or mapping tools (ELAND, TAGGER [25], RMAP [57], SEQMAP [26], SOAP [39], MAQ [38], BOWTIE [36], and ZOOM [40]) are used. An alternative option when dealing with short reads is to resort to exact set pattern matching, for which MPSCAN offers an efficient solution [49, 53].

Because of the number of reads to match, repeated application of a single pattern matching algorithm for each read would require an unaffordable computing time. Hence, practically efficient solutions involve either

1. Indexing the reads in main memory and scanning the genome only once (or a few times) for all reads (the solution chosen in MEGABLAST [64], SEQMAP [26], and MPSCAN [53])
2. First preprocessing the genome to build an index and then loading the index in memory before searching each read one after the other (the approach followed in SSAHA [48], BLAT [30], and in mapping tools like ELAND, TAGGER [25], RMAP [57], SOAP [39], MAQ [38], BOWTIE [36])

### 5.4.1 MPSCAN: An Efficient Exact Set Pattern Matching Tool for DNA/RNA Sequences

The program MPSCAN [49, 53] is an implementation of the exact multipattern BNDM with $q$-grams algorithm presented in Section 5.3.2.2. It is specialized for searching large sets of relatively short DNA/RNA patterns in large DNA/RNA sequence files, and its interface is adequate for the purpose of mapping reads; it handles file formats commonly used in biology, can search for the reverse complementary of the pattern, and so on.

Its correctness, which ensures it to yield for each read all text positions at which that read matches the text, derives from that of the multipattern BNDM with $q$-grams algorithm. The filtration efficiency depends on the parameter $q$. If we choose $q = \Theta(\log_c(rm))$, then MPSCAN is optimal on average.

For example, on an Intel Xeon CPU 5140 processor at 2.33 GHz with 8 GB main memory, when searching 4 million 27 bp reads on the 247 Mbp of human chromosome 1, MPSCAN sets the parameter $q$ to 13, uses 229 MB memory, and runs in 78 seconds.

### 5.4.2 Other Solutions for Mapping Reads

With HTS becoming more popular and the increase of sequencing capacity, the question of mapping reads on a genome sequence is a crucial issue as well as a bottleneck.

At the HTS advent, an available solution was to use ultrafast similarity search with BLAST-like programs, which were not designed for this purpose but for locally aligning sequences that differ little (*e.g.*, only because of sequencing errors). They typically were intended to align expressed sequence tags on the human genome. These programs are not adapted to short reads (below 60 bp) and because of internal limitations cannot handle millions of queries. Hence, both their sensitivity and scalability are insufficient for the mapping application with short reads [53]. However, some users still resort to these tools because, unlike mapping tools, they allow an unrestricted number of differences between the read and the genome [31]. All these tools implement a filtration strategy that requires a substring of the query sequence to match the genome either exactly [48, 64] or with at most one mismatch [30].

Since the commercialization of HTS, plenty of commercial or free mapping tools have been developed or published (*cf.* previous list); for instance, the ELAND software is provided with the Illumina Solexa sequencer. As mentioned, the goal of mapping differs with the application, but it is often to find the best match for a read—the match with the least differences and, if possible, unique. All mapping programs perform successive approximate pattern matching up to a limited number of differences. Some tools can find matches with up to four mismatches and/or indels, but generally a guarantee to find all matches (as required in the definition of approximate matching) is given only up to one or two mismatches. This limitation makes sense to speed the search and derives from the applied filtration scheme. All tools (except ZOOM) use variants of the so-called partition into exact search (PEX) filter [46], which consists of splitting the read in $k + 1$ adjacent pieces, knowing that at least one piece will match exactly when a maximum of $k$ errors are allowed. Many mapping programs make it efficient by using 2-bit encoded sequences and/or an index of the genome (*e.g.*, MAQ, ELAND, BOWTIE, RMAP, or SOAP).

The program ZOOM exploits *spaced seeds*; it requires that a subsequence of a defined form, instead of a substring, matches between the read and the genome [13,41]. The subsequence's pattern of required matching positions and wild cards is intentionally designed depending on the expected match length and maximal number of differences [35]. The advantage of spaced seeds is their capacity to handle mismatches and insertion/deletions (indels) and their increased sensitivity compared with substring-based filtration [13,41]. Their main drawback is the difficulty of seed design; ZOOM uses a conjunction of several seeds. Hence, sets of spaced seeds are designed specifically for a certain read/match length and a maximum number of allowed differences, and different sets corresponding to different parameter combinations are hard coded in ZOOM. All known formulations of the seed design problem are at least NP-hard even for a single seed [35,42,47].

### 5.4.3 Comparison of Mapping Solutions

As already mentioned, many groups have developed and/or published their own mapping tools, and all tools, except MPSCAN, implement a solution-based on approximate pattern matching. However, to date, one lacks a comparative evaluation of the sensibility of all these tools in various application frameworks. The intended application makes a difference because, for example, identifying genomic variations and multiple matching locations of a read provide useful information, whereas in transcriptomics, one usually discards multimapped reads. Probing the sensitivity and evaluating the sensitivity versus speed or memory balance is a difficult task, knowing that the programs differ in their notion of approximation (*e.g.*, with or without indels).

Here, we discuss the conclusions of a comparison on the less difficult task of exact set pattern matching. We exclude the program ELAND because it is not free for academics as well as MAQ, which does not accept parameters for searching only exact read matches.

***5.4.3.1 Speed, Memory Footprint, and Scalability.*** We compared RMAP, SEQMAP, SOAP (versions 1 and 2), ZOOM, BOWTIE, and MPSCAN for searching increasing read sets on the longest human chromosome (chromosome 1, 247 Mbp). The public input datasets contains 6.5 million 27 bp reads, and we took subsets every million reads (available on the GEO database under accession number GSM325934). At the date of this comparison, this set belongs to the largest ones in terms of number of different reads, and there is no available dataset of similar size with much longer reads (say $>$ 36 bp).

Figure 5.6 reports the running times in seconds on a logarithmic scale for searching the subsets of $1, 2, \ldots$ up to 6 and 6.5 million reads. Of course, the times do not include the index construction time for those programs that use an index, which is for example, hours for BOWTIE in the case of the complete human genome [36, Table 5].

First, all tools can handle very large read sets, and their running times remain impressive even if they degrade somehow with increasing read sets. Second, the comparison of ZOOM or MPSCAN compared with genome indexing tools like BOWTIE

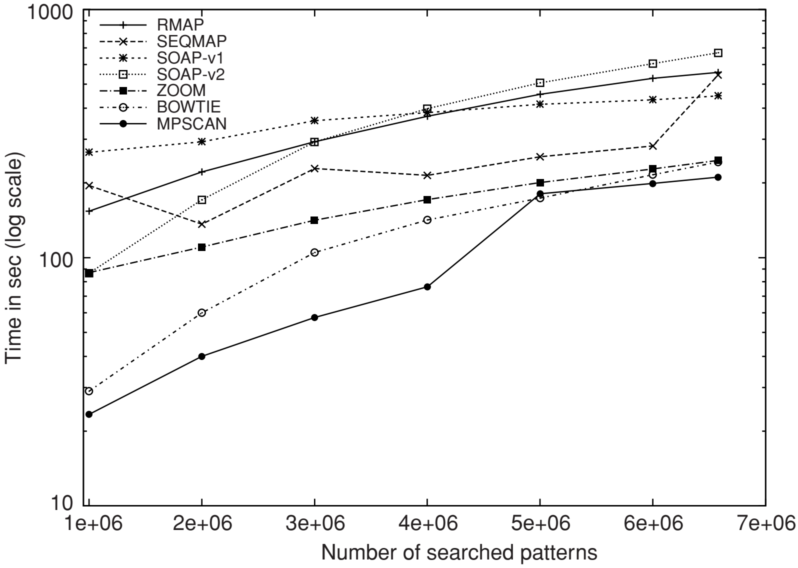


**Figure 5.6** Comparison of mapping tools: Search times of RMAP, SEQMAP, SOAP (versions 1 and 2), ZOOM, BOWTIE, and MPSCAN in seconds (log scale) for increasing subsets of 27 bp reads. All tools behave similarly and offer acceptable scalability. MPSCAN remains the most efficient of all and can be 10 times faster than tools like SEQMAP or RMAP. Times do not include the index construction time.

or SOAP shows that high performance is not bound to a genome index, at least for exact pattern matching. Knowing that the ZOOM algorithm also handles approximate matches with up to two mismatches or indels, it seems that it offers a very satisfying solution compared with BOWTIE, which is limited to mismatches and offers less guarantees. For exact pattern matching, the performance differences can be large (10 times between MPSCAN and SOAP-v2 for 2 million reads), and MPSCAN offers the fastest solution overall, even if it exploits only a 32-bit architecture. However, MPSCAN time increases more when going from 4 to 5 million reads, suggesting that for equal read length, a coarse-grain parallelization would improve its performance.

To illustrate the low memory footprint of mapping tools that do not load a genome index in RAM, we give the amount of RAM required by ZOOM, SEQMAP, and MPSCAN for searching the complete human genome with 1 million sequences for 27 bp tags. ZOOM requires 17 minutes and 0.9 gigabytes, RMAP takes 30 minutes and 0.6 gigabytes, SEQMAP performs the task in 14 minutes with 9 gigabytes, whereas MPSCAN needs < 5 minutes using 0.3 gigabytes. In contrast, the BOWTIE human genome index, which is constructed using the Burrows–Wheeler Transform, takes at least 1.4 gigabytes [36].

***5.4.3.2 Exact Pattern Matching for Read Mapping.*** The read length influences the probability of a read to map on the genome and also its probability to map once. The shorter the read, the higher the probability of mapping but the lower that of mapping once. In many applications, reads mapping at unique genomic positions are preferred. A rationale for the currently developed extension of read length is the increased probability to map to a unique genomic location. On the human genome, a length of 19 bp already brings the risk of mapping at random below 1%, and we have shown recently that it already maximizes the number of uniquely mapped reads on four real datasets [49]. Studying the sequence error position in the reads, we could show that the error probability at one sequence position increases with the position in the read for Illumina/Solexa data. Hence, an alternative to approximate mapping is to perform exact matching using only a prefix (of an adequate length) of each read.

To evaluate this, we compared the result of approximate matching with full-length reads with that of MPSCAN on read prefixes. ELAND searches the best read match up to two mismatches, whereas we ran MPSCAN to search for exact matches of read prefixes. The full-length reads are 34 bp. If one maps with MPSCAN the full-length reads, 86% remain unmapped, and 11% are uniquely mapped. With at most two mismatches, ELAND finds 14% of additional uniquely mapped reads with one or two mismatches, whereas mapping the 20 bp prefix of each read with MPSCAN allows mapping 25% of all reads at unique positions (14% more sites than with full-length reads). Hence, both approaches yield a similar output, but exact matches represent easier and more secure information than approximate matches. For the current rates of sequencing errors and read lengths, exact matching is a suitable solution for read mapping. Moreover, it allows us to estimate computationally the sequence error rate without performing control experiments (*cf.* [49] for a more in-depth presentation), which would be more difficult using approximate matching.

## 5.5 CONCLUSIONS

For pattern matching, $q$-gram-based algorithms and especially MPSCAN represent the most efficient theoretical and practical solutions to exact set pattern matching for huge pattern sets (greater than million patterns). Compared with known solutions surveyed seven years ago in [46], which were reported to handle several hundred thousands of patterns, MPSCAN provides more than an order of magnitude improvement; it allows processing at astonishing speed pattern sets of several millions reads. The second take-home message is that its filtration scheme can compete with approaches that use a text index.

Since 2005, the capacity of HTS is evolving continuously; biotechnological research and development aim at reducing the quantity of biological extract, augmenting the sequencing capacity and quality, raising the read length, and even enlarging the application fields. Despite the efforts for designing scalable and efficient mapping programs, it will remain a computational issue to let mapping solutions fit the requirements of new HTS versions. This question is complex because read lengths greater than 20 are not necessary to point out a unique position in a genome as large as that of human [49].

An interesting conclusion is that different filtration schemes achieve impressive efficiency and scalability but may be insufficient for tomorrow's needs. The abundant pattern matching literature still may contain other possible algorithms whose applications in this setup have not yet been evaluated. With the spread of multicore computers, parallelization represents another future line of research.

Finally, we left aside the problem of mapping pairs of reads. In this framework, two reads are sequenced for each targeted molecule, each at a different extremity. The reads come in pairs, and the goal of mapping is to find one matching position for each read such that the two positions are on the same chromosome and in an upper bounded vicinity. In other applications, the pair relations are unknown, and it then is required to find across the two sets of beginning and ending reads which ones constitute a pair because they map on the same chromosome not too far from another [52]. Some mapping tools like MAQ or ZOOM can solve read pair mapping efficiently, whereas a predecessor of MPSCAN has been developed and applied in the second framework [52].

## REFERENCES

1. K. Abrahamson. Generalized string matching. *SIAM J Comput*, 16(6):1039–1051, 1987.
2. A.V. Aho and M.J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun ACM*, 18(6):333–340, 1975.
3. C. Allauzen and M. Raffinot. Factor oracle of a set of words. Technical Report 99-11, Institut Gaspard-Monge, Universit de Marne-la-Vallée, 1999. (in French).
4. S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, 1997.
5. J.-M. Aury, C. Cruaud, V. Barbe, O. Rogier, S. Mangenot, G. Samson, J. Poulain, V. Anthouard, C. Scarpelli, F. Artiguenave, and P. Wincker. High quality draft sequences for prokaryotic genomes using a mix of new sequencing technologies. *BMC Genom*, 9:603, 2008.
6. R.A. Baeza-Yates. Improved string searching. *Soft—Pract Exp*, 19(3):257–271, 1989.
7. R.A. Baeza-Yates and G.H. Gonnet. A new approach to text searching. *Commun ACM*, 35(10):74–82, 1992.
8. T. Berry and S. Ravindran. A fast string matching algorithm and experimental results. *Proceedings of the Prague Stringology Club Workshop '99*, Collaborative Report DC-99-05, Czech Technical University, Prague, Czech Republic, 1999, pp. 16–28.
9. A.J. Bleasby, D. Akrigg, and T.K. Attwood. OWL – a non-redundant, composite protein sequence database. *Nucleic Acids Res*, 22(17):3574–3577, 1994.
10. N. Blow. Transcriptomics: The digital generation. *Nature*, 458:239–242, 2009.
11. R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Commun ACM*, 20(10):762–772, 1977.
12. A.P. Boyle, S. Davis, H.P. Shulha, P. Meltzer, E.H. Margulies, Z. Weng, T.S. Furey, and G.E. Crawford. High-resolution mapping and characterization of open chromatin across the genome. *Cell*, 132(2):311–322, 2008.
13. S. Burkhardt and J. Kärkkäinen. Better filtering with gapped $q$-grams. *Fundamenta Informaticae*, 56(1–2):51–70, 2003.

14. W. Chen, V. Kalscheuer, A. Tzschach, C. Menzel, R. Ullmann, M.H. Schulz, F. Erdogan, N. Li, Z. Kijas, G. Arkesteijn, I.L. Pajares, M. Goetz-Sothmann, U. Heinrich, I. Rost, A. Dufke, U. Grasshoff, B. Glaeser, M. Vingron, and H.H. Ropers. Mapping translocation breakpoints by next-generation sequencing. *Genome Res*, 18(7):1143–1149, 2008.
15. J.M. Claverie and C. Notredame. *Bioinformatics for Dummies*. Wiley Publishing, Inc., New York, NY, 2003.
16. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.
17. B. Commentz-Walter. A string matching algorithm fast on the average. *Proceedings of the 6th Colloquium on Automata, Languages and Programming (ICALP'79)*, volume 71 of *LNCS*, pages 118–132. Springer-Verlag, Graz, Austria, 1979.
18. Y.S. Dandass, S.C. Burgess, M. Lawrence, and S.M. Bridges. Accelerating string set matching in FPGA hardware for bioinformatics research. *BMC Bioinformatics*, 9(1):197, 2008.
19. B. Durian, J. Holub, H. Peltola, and J. Tarhio. Tuning BNDM with $q$-grams. *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'09)*, pages 29–37. SIAM, 2009.
20. Editor. Prepare for the deluge. *Nat Biotechnol*, 26:1099, 2008.
21. D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK, 1997.
22. R.N. Horspool. Practical fast searching in strings. *Soft—Pract Exp*, 10(6):501–506, 1980.
23. A. Hume and D. Sunday. Fast string searching. *Soft—Pract Exp*, 21(11):1221–1248, 1991.
24. D.H. Huson, D.C. Richter, S. Mitra, A.F. Auch, and S.C. Schuster. Methods for comparative metagenomics. *BMC Bioinformatics*, 10(Suppl 1):S12, 2009.
25. C. Iseli, G. Ambrosini, P. Bucher, and C.V. Jongeneel. Indexing strategies for rapid searches of short words in genome sequences. *PLoS ONE*, 2(6):e579, 2007.
26. H. Jiang and W.H. Wong. SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, 24(20):2395–2396, 2008.
27. D.S. Johnson, A. Mortazavi, R.M. Myers, and B. Wold. Genome-wide mapping of in vivo protein-DNA interactions. *Science*, 316(5830):1497–1502, 2007.
28. P. Kalsi, H. Peltola, J. Tarhio. Comparison of exact string matching algorithms for biological sequences. *Proceedings of the 2nd International Conference on Bioinformatics Research and Development (BIRD'08)*, volume 13 of Communications in Computer and Information Science, Springer-Verlag, 2008, pp. 417–426.
29. R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J Res Dev*, 31(2):249–260, 1987.
30. W.J. Kent. BLAT—the BLAST-like alignment tool. *Genome Res*, 12(4):656–664, 2002.
31. P.V. Kharchenko, M.Y. Tolstorukov, and P.J. Park. Design and analysis of ChIP-seq experiments for DNA-binding proteins. *Nat Biotechnol*, 26(12):1351–1359, 2008.
32. J.Y. Kim and J. Shawe-Taylor. Fast string matching using an $n$-gram algorithm. *Soft—Pract Exp*, 24(1):79–88, 1994.
33. J.W. Kim, E. Kim, and K. Park. Fast matching method for DNA sequences. *Proceedings of the 1st International Symposium on Combinatorics, Algorithms, Probabilitic and Experimental Methodologies (ESCAPE'07)*, volume 4614 of *LNCS*, Springer-Verlag, 2007, pp. 271–281.

34. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J Comput*, 6(2):323–350, 1977.

35. G. Kucherov, L. Noé, and M. Roytberg. Multiseed lossless filtration. *IEEE/ACM Trans Comput Biol Bioinform*, 2(1):51–61, 2005.

36. B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

37. T. Lecroq. Fast exact string matching algorithms. *Inf Process Lett*, 102(6):229–235, 2007.

38. H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–1858, 2008.

39. R. Li, Y. Li, K. Kristiansen, and J. Wang. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

40. H. Lin, Z. Zhang, M.Q. Zhang, B. Ma, and M. Li. ZOOM! Zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437, 2008.

41. B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

42. B. Ma and M. Li. On the complexity of the spaced seeds. *J Comput Syst Sci*, 73(7):1024–1034, 2007.

43. M. Margulies, M. Egholm, W.E. Altman, S. Attiya et. al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

44. G. Navarro and K. Fredriksson. Average complexity of exact and approximate multiple string matching. *Theor Comput Sci*, 321(2-3):283–290, 2004.

45. G. Navarro and M. Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM J Experimental Algorithmics*, 5(4):1–36, 2000.

46. G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical On-line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, Cambridge, UK, 2002.

47. F. Nicolas and E. Rivals. Hardness of optimal spaced seed design. *J Comput Syst Sci*, 74(5):831–849, 2008.

48. Z. Ning, A.J. Cox, and J.C. Mulikin. SSAHA: A fast Search method for large DNA databases. *Genome Res*, 11:1725–1729, 2001.

49. N. Philippe, A. Boureux, L. Bréhélin, J. Tarhio, T. Commes, and E. Rivals. Using reads to annotate the genome: Influence of length, background distribution, and sequence errors on prediction capacity. *Nucleic Acids Res*, 37(15):e104, 2009.

50. S. Rahmann. Fast large scale oligonucleotide selection using the longest common factor approach. *J Bioinformatics Comput Biol*, 1(2):343–361, 2003.

51. J. Rautio, J. Tanninen, and J. Tarhio. String matching with stopper encoding and code splitting. *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching (CPM'02)*, volume 2373 of *LNCS*, Springer-Verlag, Fakuoka, Japan, 2002, pp. 42–52.

52. E. Rivals, A. Boureux, M. Lejeune, F. Ottones, O.P. Prez, J. Tarhio, F. Pierrat, F. Ruffle, T. Commes, and J. Marti. Transcriptome annotation using tandem SAGE tags. *Nucleic Acids Res*, 35(17):e108, 2007.

53. E. Rivals, L. Salmela, P. Kiiskinen, P. Kalsi, and J. Tarhio. MPSCAN: Fast localisation of multiple reads in genomes. *Proceedings of the 9th Workshop on Algorithms in Bioinformatics (WABI'09)*, volume 5724 of *LNCS*, Springer-Verlag, 2009, pp. 246–260.

54. L. Salmela. *Improved Algorithms for String Searching Problems.* PhD dissertation, TKK Research Reports in Computer Science and Engineering A, TKK-CSE-A1/09, Helsinki University of Technology, 2009.

55. L. Salmela, J. Tarhio, and J. Kytöjoki. Multipattern string matching with $q$-grams. *ACM J Experimental Algorithmics*, 11(1.1):1–19, 2006.

56. S.S. Sheik, S.K. Aggarwal, A. Poddar, N. Balakrishnan, and K. Sekar. A FAST pattern matching algorithm. *J Chem Inform Comput Sci*, 44(4):1251–1256, 2004.

57. A.D. Smith, Z. Xuan, and M.Q. Zhang. Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics*, 9(1):128, 2008.

58. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J Mol Bio*, 147(1):195–197, 1981.

59. M. Sultan, M.H. Schulz, H. Richard, A. Magen, A. Klingenhoff, M. Scherf, M. Seifert, T. Borodina, A. Soldatov, D. Parkhomchuk, D. Schmidt, S. O'Keeffe, S. Haas, M. Vingron, H. Lehrach, and M.-L. Yaspo. A global view of gene activity and alternative splicing by deep sequencing of the human transcriptome. *Science*, 321(5891):956–960, 2008.

60. D.M. Sunday. A very fast substring search algorithm. *Commun ACM*, 33(8):132–142, 1990.

61. J. Tarhio and H. Peltola. String matching in the DNA alphabet. *Soft—Pract Exp*, 27(7):851–861, 1997.

62. R. Thathoo, A. Virmani, S.S. Lakshmi, N. Balakrishnan, and K. Sekar. TVSBS: A fast exact pattern matching algorithm for biological sequences. *Curr Sci*, 91(1):47–53, 2006.

63. S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994.

64. Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *J Comput Bio*, 7(1–2):203–214, 2000.

65. R.F. Zhu and T. Takaoka. On improving the average case of the Boyer–Moore string matching algorithm. *J Inf Process*, 10(3):173–177, 1987.

66. IUPAC-IUB Joint Commission on Biochemical Nomenclature. Nomenclature and symbolism for amino acids and peptides. *Biochem. J.*, 219:345–373, 1984.

# 6

# ALGORITHMIC ASPECTS OF ARC-ANNOTATED SEQUENCES

Guillaume Blin, Maxime Crochemore, and Stéphane Vialette

## 6.1 INTRODUCTION

Structure comparison for RNA has become a central computational problem bearing many computer science challenging questions. Indeed, RNA secondary structure comparison is essential for (i) identification of highly conserved structures during evolution (which always cannot be detected in the primary sequence because it is often unpreserved), which suggests a significant common function for the studied RNA molecules, (ii) RNA classification of various species (phylogeny), (iii) RNA folding prediction by considering a set of already known secondary structures, and (iv) identification of a consensus structure and consequently of a common role for molecules.

From an algorithmic point of view, RNA structure comparison first was considered in the framework of ordered trees [21]. More recently, it also has been considered in the framework of *arc-annotated sequences* [10]. An arc-annotated sequence is a pair $(S, P)$, where $S$ is a sequence of RNA bases and $P$ represents hydrogen bonds between pairs of elements of $S$. From a purely combinatorial point of view, arc-annotated sequences are a natural extension of simple sequences. However, using arcs for modeling nonsequential information together with restrictions on the relative positioning of arcs allow for varying restrictions on the structure of arc-annotated sequences.

Different pattern matching and motif search problems have been considered in the context of arc-annotated sequences among which we can mention the longest

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

arc-annotated subsequence (LAPCS) problem, the arc preserving subsequence (APS) problem, the maximum arc-preserving common subsequence (MAPCS) problem, and the edit-distance for arc-annotated sequence (EDIT) problem. This chapter is devoted to presenting algorithmic results for these arc-annotated problems.

This chapter is organized as follows. We present basic definitions in Section 6.2. Section 6.3 is devoted to the problem of finding a LAPCS between two arc-annotated sequences, whereas we consider in Section 6.4 the restriction of deciding whether an arc-annotated sequence occurs in another arc-annotated sequence, the so-called APS problem. Section 6.5 is concerned with some variants of the longest arc-preserving common subsequence problem. Section 6.6 is devoted to computing the edit distance between two arc-annotated sequences.

## 6.2 PRELIMINARIES

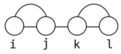
### 6.2.1 Arc-Annotated Sequences

Given a finite alphabet $\Sigma$, an arc-annotated sequence is defined by a pair $(S, P)$, where $S$ is a string of $\Sigma^*$ and $P$ is a set of arcs connecting pairs of characters of $S$. The set $P$ usually is represented by set of pairs of positions in $S$. Characters that are not incident to any arc are called *free*.

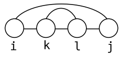
In the context of RNA structures, we have $\Sigma = \{A,C,G,U\}$, and $S$ and $P$ represent the nucleotide sequence and the hydrogen bonds of the RNA structure, respectively. Characters in $S$ thus often are referred to as *bases*.

Relative positioning of arcs is of particular importance for arc-annotated sequences and is described completely by three binary relations. Let $p_1 = (i, j)$ and $p_2 = (k, l)$ be two arcs in $P$ that do not share a vertex. Define

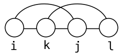
*the precedence* relation $(<)$ – $p_1 < p_2$ if $i < j < k < l$



*the embedding* relation $(\sqsubset)$ – $p_1 \sqsubset p_2$ if $i < k < l < j$



*the crossing* relation $(\between)$ – $p_1 \between p_2$ if $i < k < j < l$

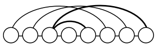


Using arcs for modeling nonsequential information together with these relations allows for varying restrictions on the complexity of arc-annotated sequences.

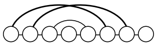
### 6.2.2 Hierarchy

The following five levels of arc structure initially have been considered in the foundation work of Evans [9]:

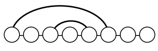
Unlimited (Unlim) – no restriction at all



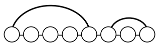
Crossing (Cros) – there is no character incident to more than one arc



NESTED (NEST) – there is no character incident to more than one arc and no arcs are crossing



CHAIN (CHAIN) – there is no character incident to more than one arc, no arcs are crossing and no arc embedded into another



PLAIN – there is no arc

The induced hierarchy is described by the following chain of inclusion:

$$\text{PLAIN} \subset \text{CHAIN} \subset \text{NESTED} \subset \text{CROSSING} \subset \text{UNLIMITED}.$$

### 6.2.3 Refined Hierarchy

In [13], Guignon *et al.* extended the aforementioned hierarchy by introducing a new refinement of the NESTED level called STEM; no character is incident to more than one arc, and given any two arcs, one is embedded in the other.

For providing a unified framework and as a next step toward a better understanding of the inner complexity of the problems related to arc-annotated sequences, Blin *et al.* [4] proposed to refine the hierarchy further following the example of Vialette [22, 23] in the context of 2-*intervals* (a simple abstract structure for modeling RNA secondary structures). The refinement consists of splitting those models of arc-annotated sequences into more precise relations between arcs, taking advantage of the combinatorics induced by the relations $<$, $\sqsubset$, and $\between$.

Two arcs $p_1$ and $p_2$ are $R$-comparable for some $R \in \{<, \sqsubset, \between\}$ if $p_1 R p_2$ or $p_2 R p_1$. Let $P$ be a set of arcs and $\mathcal{R}$ be a nonempty subset of $\{<, \sqsubset, \between\}$. The set $P$ is $\mathcal{R}$-*comparable* if any two distinct arcs of $P$ are $R$-comparable for some $R \in \mathcal{R}$. An arc-annotated sequence $(S, P)$ is an $\mathcal{R}$-arc-annotated sequence for a nonempty subset $\mathcal{R} \subseteq \{<, \sqsubset, \between\}$ if $P$ is $\mathcal{R}$-comparable. By abuse of notation, we will write $R = \emptyset$ in case $P = \emptyset$.

As a straightforward illustration of these definitions, most levels in the classical hierarchy can be expressed in terms of a combination of the three relations; PLAIN is described fully by $\mathcal{R} = \emptyset$, CHAIN is described fully by $\mathcal{R} = \{<\}$, STEM is described fully by $\mathcal{R} = \{\sqsubset\}$, NESTED is described fully by $\mathcal{R} = \{<, \sqsubset\}$, and CROSSING is described fully by $\mathcal{R} = \{<, \sqsubset, \between\}$. The key point is to observe that this refinement allows us to consider new levels for arc-annotated sequences, namely $\mathcal{R} = \{\between\}$, $\mathcal{R} = \{<, \between\}$, and $\mathcal{R} = \{\sqsubset, \between\}$.

### 6.2.4 Alignment

Given two sequences $S$ and $T$ on a common alphabet $\Sigma$, we define an *alignment* of $S$ and $T$ as a pair of sequences $(S', T')$ built from $S$ and $T$ on $\Sigma \cup \{-\}$ ($-$ is usually referred to as a *gap*) such that (i) $|S'| = |T'|$, (ii) for any $1 \leq i \leq |S'|$, either $S'[i] = T'[i] \neq -$ or exactly one of $S'[i]$ and $T'[i]$ is a gap, and (iii) removing the gaps from $S'$ or $T'$ yields $S$ or $T$, respectively).

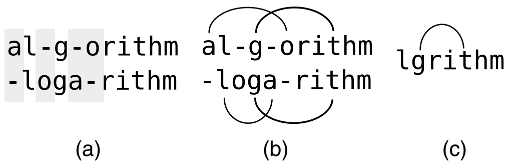


**Figure 6.1** Illustration of a) sequences of alignment leading to a common subsequence which is "lgrtihm," b) an arc-preserving alignment of two arc-annotated sequences, and c) the resulting common arc-annotated subsequence.

Let $(S', T')$ be an alignment of $S$ and $T$. For any $1 \leq i \leq |S'|$ such that $S'[i] \neq -$, character $S'[i]$ is said to be *aligned* with character $T'[i]$ if $T'[i] \neq -$ and *deleted* otherwise. Similarly, For any $1 \leq i \leq |T'|$ such that $T'[i] \neq -$, character $T'[i]$ is said to be *aligned* with character $S'[i]$ if $S'[i] \neq -$ and *inserted* otherwise. An illustration is given in Figure 6.1.

An alignment $(S', T')$ of two arc-annotated sequences $(S, P)$ and $(T, Q)$ is *arc-preserving* if the arcs induced by $(S', T')$ are preserved (*i.e.*, the arcs induced by the aligned bases are preserved). In this context, the notion of common subsequence is extended by including the common arcs—that is, the arcs that have been preserved by the alignment.

### 6.2.5 Edit Operations

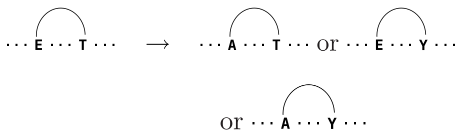
Following the example of stringology, when comparing two arc-annotated sequences $(S, P)$ and $(T, Q)$, instead of computing an alignment, one might consider a set of edit operations (together with their associate costs) that alter arc-annotated sequences and seek for a minimal cost sequence according to these operations that leads from $(S, P)$ to $(T, Q)$.

Formally, given a set of edit operations $\mathcal{E}$ and two arc-annotated sequences $(S, P)$ and $(T, Q)$, an *edit-script* from $(S, P)$ to $(T, Q)$ refers to a series of nonoriented operations of $\mathcal{E}$ transforming $(S, P)$ into $(T, Q)$. The *cost of an edit-script* from $(S, P)$ to $(T, Q)$, denoted $cost((S, P), (T, Q), \mathcal{E})$, is the sum of the costs of all operations involved in the edit-script. The *edit-distance* between $(S, P)$ and $(T, Q)$ is the minimum cost of an edit-script from $(S, P)$ to $(T, Q)$.

The classical approach is to consider a subset of the operations introduced in [15], which can be divided into the following groups:

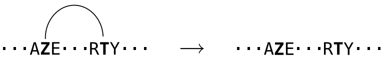
**Substitution operations**, inducing the renaming of characters in the arc-annotated sequence:

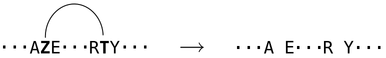
| | | | | |
|---|---|---|---|---|
| match | $(w_m : \Sigma \to \mathbb{R})$ | $\cdots \mathtt{R} \cdots$ | $\longrightarrow$ | $\cdots \mathtt{R} \cdots$ |
| mismatch | $(w_m : \Sigma \to \mathbb{R})$ | $\cdots \mathtt{R} \cdots$ | $\longrightarrow$ | $\cdots \mathtt{W} \cdots$ |
| arc-match | $(w_{am} : \Sigma^4 \to \mathbb{R})$ | $\cdots \overset{\frown}{\mathtt{E} \cdots \mathtt{T}} \cdots$ | $\longrightarrow$ | $\cdots \overset{\frown}{\mathtt{E} \cdots \mathtt{T}} \cdots$ |

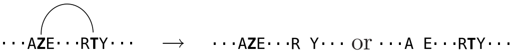
arc-mismatch ($w_{am} : \Sigma^4 \rightarrow \mathbb{R}$) 

**Deletion operations**, inducing the deletion of characters and/or of arcs in the arc-annotated sequence:

deletion ($w_d : \Sigma \rightarrow \mathbb{R}$) ···ERY··· → ···E Y···

arc-breaking ($w_b : \Sigma^4 \rightarrow \mathbb{R}$) 

arc-removing ($w_r : \Sigma^2 \rightarrow \mathbb{R}$) 

arc-altering ($w_a : \Sigma^3 \rightarrow \mathbb{R}$) 

## 6.3 LONGEST ARC-PRESERVING COMMON SUBSEQUENCE

### 6.3.1 Definition

The LAPCS problem has been introduced by Evans [9] and is defined as follows: given two arc-annotated sequences $(S, P)$ and $(T, Q)$, find an arc-preserving common subsequence of maximal length. The computational complexity of the LAPCS problem has been studied in [9, 10, 17, 18, 14, 7], and the main results are summarized in Tables 6.1, 6.2 and 6.3.

**Table 6.1 LAPCS classical complexity with $n = |S|$ and $m = |T|$**

| $A \times B$ | LAPCS |
|---|---|
| Stem × Stem | NP-complete – Blin *et al.* [7] |
| Chain × Chain<br>Nest × Chain | $O(nm^3)$ – Jiang *et al.* [17] |
| Nest × Nest | NP-complete even for unary, c-fragment (with $c > 2$) and c-diagonal (with $c > 1$) – Jiang [18] |
| Cros × Chain<br>Cros × Nest | NP-complete – Evans [9] |
| Cros × Cros | NP-complete – Evans [9] but polynomial-time solvable for 1-fragment LAPCS(Crossing, Crossing) and 0-diagonal LAPCS(Crossing, Crossing) [18] |
| Unlim × Chain<br>Unlim × Nest<br>Unlim × Cros<br>Unlim × Unlim | NP-complete – Evans [9] |

**Table 6.2 LAPCS parameterized complexity with $n = |S|$ and $m = |T|$**

| $A \times B$ | LAPCS |
|---|---|
| STEM × STEM | FPT when parameterized by the number of deletions – Alber *et al.* [1] |
| NEST × CHAIN<br>NEST × NEST | FPT when parameterized by the bandwidth or the nesting depth – Evans [9], FPT when parameterized by the number of deletions – Alber *et al.* [1] |
| CROS × CHAIN<br>CROS × NEST | FPT when parameterized by the bandwidth or the cutwidth – Evans [9], Jiang *et al.* [16] |
| CROS × CROS | W[1]-complete and FPT when parameterized by the bandwidth or the cutwidth – Evans [9], Fixed Parameter Tractable (FPT) when parameterized by the desired common subsequence length – Alber *et al.* [1] |
| UNLIM × CHAIN<br>UNLIM × NEST<br>UNLIM × CROS<br>UNLIM × UNLIM | W[1]-complete – Evans [9] |

In the sequel, we use the notation LAPCS($A$, $B$) to represent the LAPCS problem in which the arc structure of $S$ (or $T$) – namely $P$ (or $Q$) – is of level $A$ (or $B$), respectively.

### 6.3.2 Classical Complexity

In [9], Evans proved that LAPCS(CHAIN, CHAIN) is polynomial-time solvable, whereas both LAPCS(UNLIMITED, PLAIN) and LAPCS(CROSSING, PLAIN) are nondeterministic polynomial NP-complete (reductions from independent set). In [18], Lin *et al.* proved that LAPCS(NESTED, NESTED) is NP-complete (reduction from independent set). Complementing these results, Jiang *et al.* [17] designed an $O(nm^3)$ time algorithm for LAPCS(NESTED, CHAIN) and LAPCS(CHAIN, CHAIN). Recently,

**Table 6.3 LAPCS approximability**

| $A \times B$ | LAPCS |
|---|---|
| NEST × CHAIN<br>NEST × NEST | 2-approximable – Jiang *et al.* [16], PTAS for c-fragmented and c-diagonal cases [18] |
| CROS × CHAIN<br>CROS × NEST<br>CROS × CROS | MaxSNP-hard, 2-approximable – Jiang *et al.* [16] |
| UNLIM × CHAIN<br>UNLIM × NEST<br>UNLIM × CROS<br>UNLIM × UNLIM | Cannot be approximated within ratio $n^\epsilon$ for any $\epsilon \in (0, \frac{1}{4})$ – [16] |

Blin *et al.* [7] proved that LAPCS(Stem, Stem) is NP-complete (reduction from 3-SAT).

Lin *et al.* further investigated this last problem by studying restricted cases, namely, $c$-fragmented, $c$-diagonal, and unary LAPCS(Nested, Nested). Given two arc-annotated sequences that are divided into fragments of lengths exactly $c$ (the last fragment can have a length less than $c$), the *c-fragment* LAPCS problem with $c \geq 1$, is defined as the classical LAPCS problem with the extra constraint that the allowed matches are those between fragments at the same location [14]. The *c-diagonal* LAPCS problem with $c \geq 0$ is an extension of $c$-fragment LAPCS, where character $S[i]$ is allowed only to match a character in the range $T[i - c, i + c]$. Lin *et al.* [18] showed the NP-hardnessof the c-fragment (with $c > 2$) and c-diagonal (with $c > 1$) LAPCS (Nested, Nested) problem. They also proved that the 1-fragment LAPCS(Crossing, Crossing) and 0-diagonal LAPCS(Crossing, Crossing) are solvable in time $O(n)$.

### 6.3.3 Parameterized Complexity

Considering the parameter $l$ as being the desired length of common subsequence, Evans [9], using one of the previously-mentioned reductions for LAPCS (Unlimited, Plain) and providing a reduction from Clique to LAPCS(Crossing, Crossing), roved that both LAPCS(Unlimited, Plain) and LAPCS(Crossing, Crossing) are W[1]-complete when parameterized by $l$. Moreover, Evans proved in [10] that although LAPCS(Crossing, Crossing) is W[1]-complete, then the problem becomes fixed-parameter tractable when parameterized by the arc cutwidth. The *arc cutwidth* [10] of an arc-annotated sequence is defined as the maximal number of arcs that cross or end at any arbitrary position of the sequence. If both sequences have their cutwidth bounded by some $k$, then the problem, as shown by Evans, can be solved in $O(9^k nm)$ time, where $|S| = n$ and $|T| = m$. Evans also investigated the parameterized complexity of the problem considering two other parameters: the bandwidth and the nesting depth. The *bandwidth d* of an arc-annotated sequence $(S, P)$ is defined by $\max_{(i,j)\in P}\{|j - i|\}$ and its nesting depth $s$ is equal to $\max\{|P'|\}$, where $P' \subseteq P$ such that for all $(i, j) \in P'$, $(k, l) \in P'$ does not exist with $i < k < j < l$ or $i < j < k < l$. Evans showed that if both sequences have their nesting depth bounded by some $s$, then LAPCS(Nested, Nested) can be solved in $O(s^2 4^s nm)$ time, where $|S| = n$ and $|T| = m$. In case the arcs do not share endpoints, both cutwidth and nesting depth are always no more than bandwidth. Thus, Evans, was able to extend the previously mentioned results to the parameter $d$. Finally, one has to observe that if the complexity of the arc structure is bounded by a logarithm of the maximal sequence length $n$, then LAPCS can be solved in $O(n^2 m)$ time even for Crossing type arc structures. Moreover, because the cutwidth is equal to one for LAPCS(Chain, Chain), one can use the algorithm for LAPCS(Crossing, Crossing) to solve this problem in $O(nm)$ time.

Considering LAPCS(Nested, Nested), Alber *et al.* [1] designed an algorithm that determines in time $O(3.31^{k_1+k_2} n)$ whether an arc-preserving common subsequence can be obtained by deleting (together with incident arcs) $k_1$ characters from $S$ and $k_2$ from $T$, thereby proving that LAPCS(NESTED, NESTED) is fixed-parameter tractable when parameterized by the number of deletions. Finally, Alber *et al.* [1] showed that $c$-fragment LAPCS(CROSSING, CROSSING) and $c$-diagonal LAPCS(CROSSING, CROSSING) parameterized by the length $l$ of the desired common subsequence are solvable in $O((B+1)^l B + c^3 n)$ time, with $B = c^2 + 2c - 1$ and $B = 2c^2 + 7c + 2$, respectively.

### 6.3.4 Approximability

Jiang *et al.* in [16] proved that LAPCS(CROSSING, CROSSING) admits a simple 2-approximation algorithm running in $O(nm)$ time, whereas LAPCS(UNLIMITED, PLAIN) cannot be approximated within ratio $n^\epsilon$ for any $\epsilon \in (0, \frac{1}{4})$, where $n$ denotes the length of the longest input sequence. In the same paper, they proved that LAPCS(CROSSING, PLAIN) is MaxSNP-hard, thereby excluding a polynomial-time approximation scheme (PTAS). Jiang *et al.* [18] proved that both $c$-fragmented and $c$-diagonal LAPCS(NESTED, NESTED) have a PTAS. They also give a $\frac{4}{3}$-approximation algorithm for the unary LAPCS(NESTED, NESTED) problem.

## 6.4 ARC-PRESERVING SUBSEQUENCE

### 6.4.1 Definition

The APS problem is a decision problem derived from LAPCS. Given two arc-annotated sequences $(S, P)$ and $(T, Q)$ the APS problem asks whether $(T, Q)$ is the LAPCS of $(S, P)$ and $(T, Q)$ (*i.e.*, $(T, Q)$ is an arc-preserving subsequence of $(S, P)$). The computational complexity of the APS problem has been studied in [9, 11, 12, 14, 5, 4], and the main results are summarized in Tables 6.4 and 6.5.

In the following, we use the notation APS$(A, B)$ to represent the APS problem in which the arc structure of $S$ (or $T$) – namely $P$ (or $Q$) – is of level $A$ (or $B$), respectively.

**Table 6.4 APS classical complexity with $n = |S|$ and $m = |T|$**

| $A \times B$ | APS |
|---|---|
| CHAIN × CHAIN<br>NEST × CHAIN<br>NEST × NEST | $O(nm)$ – Guo *et al.* [11, 12] |
| CROS × PLAIN | NP-complete – Blin *et al.* [5, 4] |
| CROS × CHAIN<br>CROS × NEST | NP-complete – Guo *et al.* [11, 12] |
| CROS × CROS<br>UNLIM × CHAIN<br>UNLIM × NEST<br>UNLIM × CROS<br>UNLIM × UNLIM | NP-complete – Evans [9] |

**Table 6.5 APS classical refined complexity where $n = |S|$ and $m = |T|$**

| $A \times B$ | APS |
|---|---|
| $\{<\} \times \emptyset$ | $O(n+m)$ Guo *et al.* [11] |
| $\{<\} \times \{<\}$<br>$\{\sqsubset\} \times *$<br>$\{<, \sqsubset\} \times *$ | Guo *et al.* [11, 12] |
| $\{\between\} \times \emptyset$<br>$\{\between\} \times \{\between\}$ | $O(nm^2)$ – Blin *et al.* [5, 4] |
| $\{<, \between\} \times *$<br>$\{\sqsubset, \between\} \times *$<br>$\{<, \sqsubset, \between\} \times *$ | NP-complete – Blin *et al.* [5, 4], Guo [14], Evans [9] |

### 6.4.2 Classical Complexity

Guo proved in [14] that the APS(Crossing, Chain) problem is NP-hard. Guo *et al.* observed in [11, 12] that the NP-completenessof the APS(Crossing, Crossing) and APS(Unlimited, Plain) easily follows from LAPCS Evans' work [9]. Furthermore, they gave an $O(nm)$ time algorithm for the APS(Nested, Nested) problem. This algorithm can be applied to easier problems such as APS(Nested, Chain), APS(Nested, Plain), APS(Chain, Chain), and APS(Chain, Plain). Finally, Guo *et al.* mentioned in [11, 12] that APS(Chain, Plain) can be solved in $O(n+m)$ time. Finally, Blin *et al.* [5, 4] proved that APS(Crossing, Plain) is NP-complete.

### 6.4.3 Classical Complexity for the Refined Hierarchy

In analyzing the computational complexity of a problem, we often are trying to define the precise boundary between the polynomial and the NP-complete cases. Therefore, as another step toward establishing the precise complexity landscape of the APS problem, it is of particular interest to refine the classical complexity levels of the APS problem to define precisely what makes the problem hard. To this aim, Blin *et al.* [5, 4] used the framework introduced by Vialette [23] in the context of two-intervals. As a consequence, the number of complexity levels rises from four (not taking into account the Unlimited case) to eight.

On the positive side, Gramm *et al.* have shown that APS(Nested, Nested) is solvable in $O(nm)$ time [11, 12]. Another way of stating this result is to say that APS($\{<, \sqsubset\}, \{<, \sqsubset\}$) is solvable in $O(mn)$ time. According to the properties of the refined hierarchy, that result may be summarized by saying that APS($R_1$, $R_2$) for any compatible $R_1$ and $R_2$ such that $\between \notin R_1$ and $\between \notin R_2$ is polynomial-time solvable.

Conversely, the NP-completenessof APS(Crossing, Crossing) has been proved by Evans [9]. A simple reading shows that her proof is actually concerned with $\{<, \sqsubset, \between\}$-arc-annotated sequences and, hence, actually proves that APS($\{<, \sqsubset, \between\}$, $\{<, \sqsubset, \between\}$) is NP-complete. Similarly, in proving that APS(Crossing, Chain) is NP-complete [14], Guo actually proved that APS($\{<, \sqsubset, \between\}, \{<\}$) is NP-complete.

Therefore, both $\text{APS}(\{<, \sqsubset, \between\}, \{<, \sqsubset\})$ and $\text{APS}(\{<, \sqsubset, \between\}, \{<, \between\})$ are NP-complete.

In [5, 4], Blin *et al.* proved that both $\text{APS}(\{\sqsubset, \between\}, \emptyset)$ and $\text{APS}(\{<, \between\}, \emptyset)$ are NP-complete. They also gave a polynomial time algorithm to show that both $\text{APS}(\{\between\}, \{\between\})$ and $\text{APS}(\{\between\}, \emptyset)$ problems can be solved in $O(nm^2)$ time. In other words, they proved that the relation $\between$ alone does not imply hardness.

### 6.4.4 Open Problems

The refinement suggested by Blin *et al.* in [5, 4] shows that an APS problem becomes hard when one considers sequences containing $\{\between, R\}$-comparable for some $R \subseteq \{<, \sqsubset, \between\}$. Therefore, crossing arcs alone do not imply APS hardness. It is of course a challenging problem to explore further the complexity of the APS problem, especially the parameterized views, by considering additional parameters such as the cutwidth or the depth of the arc structures.

## 6.5 MAXIMUM ARC-PRESERVING COMMON SUBSEQUENCE

### 6.5.1 Definition

The MAPCS problem was introduced by Blin *et al.* [3] as an intermediate model for comparing arc-annotated sequences – lying between LAPCS and the EDIT (see Section 6.6). The MAPCS problem is defined as follows: given two arc-annotated sequences $(S, P)$ and $(T, Q)$, and two functions $f_b : \Sigma \to \mathbb{N}^*$ and $f_a : \Sigma^2 \to \mathbb{N}^*$, find a common arc-annotated subsequence $(U, R)$ that maximizes the following score function: $\sum_{c \in U} f_b(c) + \sum_{(c_1, c_2) \in R} f_a(c_1, c_2)$. In other words, the MAPCS problem seeks to find a common subsequence whose score takes into account both the number of bases and arcs. The computational complexity of the MAPCS problem was determined fully in [3], and the main results are summarized in Table 6.6.

**Table 6.6 MAPCS* and MAPCS classical complexity for $n = |S|$ and $m = |T|$**

| $A \times B$ | MAPCS$^*$ | MAPCS |
|---|---|---|
| Chain × Chain | $O(nm)$ | $O(nm)$ |
| Nest × Chain | $O(n^2 m)$ | $O(nm^3)$ |
| Nest × Nest | $O(n^2 m^2)$ | NP-complete |
| Cros × Chain<br>Cros × Nest | $O(n^4 \log^3 n)$ | |
| Cros × Cros | NP-complete | |
| Unlim × Chain<br>Unlim × Nest | $O(n^4 \log^3 n)$ | |
| Unlim × Cros<br>Unlim × Unlim | NP-complete | |

In the following, we use the notation MAPCS($A$, $B$) to represent the MAPCS problem where the arc structure of $S$ (or $T$) – namely $P$ (or $Q$) – is of level $A$ (or $B$), respectively.

### 6.5.2 Classical Complexity

In [3], Blin *et al.* first investigated two special cases of MAPCS, namely when one allows function $f_a$ or $f_b$ to return to zero. They easily noticed that $f_a(x, y) = 0$ for all $(x, y) \in \Sigma^2$ reduces to the LAPCS problems. They investigate the case $f_b(x) = 0$ for all $x \in \Sigma$, problems called MAPCS$*$, and proved that MAPCS$*$(CHAIN,CHAIN) can be solved in $O(nm)$ time, MAPCS$*$(NESTED,NESTED) in $O(n^2m^2)$ time, MAPCS$*$(NESTED,CHAIN) in $O(nm^2)$ time, and MAPCS$*$(UNLIMITED,NESTED) in $O(n^4 \log^3 n)$ time, where $n = |S|$ and $m = |T|$. They also proved that MAPCS$*$(CROSSING,CROSSING) is NP-complete by providing a reduction from Clique.

They also completely investigated the complexity of MAPCS by proposing an $O(nm)$ and $O(nm^3)$ time algorithm for MAPCS (CHAIN,CHAIN) and MAPCS (NEST,CHAIN) respectively, and by proving that both MAPCS (NESTED,NESTED) and MAPCS (CROSSING,PLAIN) are NP-complete.

### 6.5.3 Open Problems

As far as we know, neither the parameterized complexity nor the approximability of MAPCS have been studied (except $f_a$ always returns to zero because it corresponds to the LAPCS problems and inherits all its complexity results).

## 6.6 EDIT DISTANCE

### 6.6.1 Definition

Given two arc-annotated sequences, the EDIT problem is to find the edit-distance between ($S$, $P$) and ($T$, $Q$). It has been studied extensively [15, 19, 13, 8, 6, 3, 2, 7] (see Table 6.7 and 6.8).

### 6.6.2 Classical Complexity

Lin *et al.* proved in [19] that the problem EDIT (CROSSING, PLAIN) is NP-complete, and gave a (polynomial time) dynamic programming algorithm for the EDIT (NESTED, PLAIN) problem. Sankoff [20] previously had solved EDIT (PLAIN, PLAIN).

Blin *et al.* [8] proved that the LAPCS problem actually can be seen as a very specific case of the EDIT problem. More precisely, any edit script of minimum cost goes through a common subsequence of optimal score. This means that finding one allows finding the other. Thus, LAPCS can be seen as a particular case of EDIT in which the cost system for edit operations is the following: $w_r = 2w_d = 2w_a$, and

**Table 6.7 EDIT classical complexity for $n = |S|$ and $m = |T|$**

| $A \times B$ | EDIT |
|---|---|
| STEM × STEM | NP-complete – Blin *et al.* [7] |
| CHAIN × CHAIN<br>NEST × CHAIN | $O(nm^3)$ – Lin *et al.* [19] |
| NEST × NEST | NP-complete – Jiang *et al.* [15] and Blin *et al.* [6] |
| CROS × CHAIN<br>CROS × NEST<br>CROS × CROS<br>UNLIM × CHAIN<br>UNLIM × NEST<br>UNLIM × CROS<br>UNLIM × UNLIM | NP-complete – Lin *et al.* [19] |

all substitution operations and arc-breakings are prohibited by an arbitrary high cost. The main idea is to penalize deletion operations proportionally to the number of bases that are deleted. This last result proved that the complexity of EDIT (NESTED, NESTED) simply follows from the complexity of LAPCS(NESTED, NESTED). These results were extended in [6] where the authors showed that only a very restricted number of instances of EDIT (NESTED, NESTED) were shown to be NP-complete and that the corresponding cost system needed to satisfy restrictions that can be discussed biologically. Therefore, as another step toward establishing the precise complexity landscape of the EDIT problem, they considered a more accurate class of instances—but not overlapping with the one used in the proof from LAPCS – for determining more precisely what makes the problem hard.

The authors want to point out another interesting result from Blin *et al.* [8]—namely a unifying framework to express comparison of arc-annotated sequences called ALIGN. Indeed, Blin *et al.* showed that this hierarchy brings together most comparison models for arc-annotated sequence and leads to the introduction of new comparison models that are biologically relevant. In particular, they proposed two polynomial time algorithms for the problem of comparing two NESTED arc-annotated

**Table 6.8 EDIT approximability for $n = |S|$ and $m = |T|$**

| $A \times B$ | EDIT |
|---|---|
| NEST × NEST | $\max\left\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\right\}$-approximable – Lin *et al.* [19] |
| CROS × CHAIN<br>CROS × NEST<br>CROS × CROS<br>UNLIM × CHAIN<br>UNLIM × NEST<br>UNLIM × CROS<br>UNLIM × UNLIM | MaxSNP-hard – Lin *et al.* [19] |

sequences, whereas corresponding algorithms considering the same set of edit operations in other formalisms are not polynomial time solvable. Because it did not only rely on arc-annotated sequences, we decided not to include it in this contribution.

In [13], Guignon *et al.* introduced the notion of *conservative edit distance and mapping* between two RNA stem-loops to design a polynomial-time algorithm for comparing general secondary RNA structures using the full set of biological edit operations introduced in [15]. This algorithm is based on a decomposition in stem-loop-like substructures that are pairwised compared and used to compare complete RNA secondary structures. As mentioned in [13], although in the very restrictive case of conservative distance and mapping, the computation of the general edit distance is polynomial-time solvable, it was not known whether the general (*i.e.*, not conservative) edit distance between two stem-loops also can be computed in polynomial-time. In [7], Blin *et al.* proved that the general edit distance is indeed NP-complete.

### 6.6.3 Approximability

Lin *et al.* proved in [19] that the problem EDIT (CROSSING, PLAIN) is MaxSNP-hard. They also showed that EDIT (NESTED,NESTED) has a polynomial-time approximation algorithm with ratio $\beta = \max\{\frac{2w_a}{w_b+w_r}, \frac{w_b+w_r}{2w_a}\}$.

### 6.6.4 Open Problems

The approximation ratio of EDIT (NESTED,NESTED) depends on the respective values of the parameters $w_a$, $w_b$, and $w_r$. An interesting question is whether a polynomial-time algorithm exists with *constant* approximation ratio.

## REFERENCES

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Towards optimally solving the longest common subsequence problem for sequences with nested arc annotations in linear time. In A. Apostolico and M. Takeda, editors, *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching (CPM), Fukuoka, Japan*, volume 2373 of *Lecture Notes in Computer Science*, Springer, New York, 2002, pp. 99–114.
2. G. Blin, A. Denise, S. Dulucq, C. Herrbach, and H. Touzet. Alignment of RNA structures. *IEEE/ACM Trans Comput Biol Bioinform*, 2008. To appear.
3. G. Blin, G. Fertin, G. Herry, and S. Vialette. Comparing RNA structures: Towards an intermediate model between the edit and the lapcs problems. In Marie-France Sagot and Maria Emilia Telles Walter, editors, *1st Brazilian Symposium on Bioinformatics (BSB'07)*, volume 4643 of *Lecture Notes in Bioinformatics*, Angra dos Reis, Brazil, Springer-Verlag, New York, 2007, pp. 101–112.
4. G. Blin, G. Fertin, R. Rizzi, and S. Vialette. What makes the arc-preserving subsequence problem hard? *Trans Comput Syst Biol*, 2:1–36, 2005.
5. G. Blin, G. Fertin, R. Rizzi, and S. Vialette. What makes the arc-preserving subsequence problem hard? In *Proceedings of the International Workshop on Bioinformatics Research*

*and Applications (IWBRA)*, volume 3515 of *Lecture Notes in Computer Science*, 2005, pp. 860–868.

6. G. Blin, G. Fertin, I. Rusu, and C. Sinoquet. Extending the hardness of RNA secondary structures. In Bo Chen, Mike Paterson, and Guochuan Zhang, editors, *1st International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE'07)*, volume 4614 of *LNCS*, Hangzhou, China, Springer-Verlag, New York, 2007, pp. 140–151.
7. G. Blin, S. Hamel, and S. Vialette. Comparing RNA structures using a full set of biologically relevant edit operations is intractable. Technical Report, Université Paris Est, I.G.M., 2008. Electronic version: Preprint arXiv:0812.3946.
8. G. Blin and H. Touzet. How to compare arc-annotated sequences: The alignment hierarchy. In Fabio Crestani, Paolo Ferragina, and Mark Sanderson, editors, *13th String Processing and Information Retrieval (SPIRE'06)*, volume 4209 of *LNCS*, Glasgow, UK, Springer-Verlag, New York, 2006, pp. 291–303.
9. P. Evans. *Algorithms and Complexity for Annotated Sequences Analysis*. PhD dissertation, University of Victoria, Melborne, Australia, 1999.
10. P. Evans. Finding common subsequences with arcs and pseudoknots. In M. Crochemore and M. Paterson, editors, *Proceedings of the 10th Annual Symposium Combinatorial Pattern Matching (CPM), Warwick University, UK*, volume 1645 of *Lecture Notes in Computer Science*, Springer, New York, 1999, pp. 270–280.
11. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. In M. Agrawal and A. Seth, editors, *Proceedings of the 22nd Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Kanpur, India*, volume 2556 of *Lecture Notes in Computer Science*, 2002, pp. 182–193.
12. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. *ACM Trans Algorithm*, 2(1):44–65, 2006.
13. V. Guignon, C. Chauve, and S. Hamel. An edit distance between RNA stem-loops. In Mariano P. Consens and Gonzalo Navarro, editors, *12th International Conference SPIRE*, volume 3772 of *Lecture Notes in Computer Science*, 2005, pp. 335–347.
14. J. Guo. Exact Algorithms for the Longest Common Subsequence Problem for Arc-Annotated Sequences. Master's thesis, Univeristy of Tübingen, Tübingen, Germany, 2002.
15. T. Jiang, G. Lin, B. Ma, and K. Zhang. A general edit distance between RNA structures. *J Comput Biol*, 9(2):371–388, 2002.
16. T. Jiang, G. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. *J Discrete Algorithm*, 2(2):257–270, 2004.
17. T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In R. Giancarlo and D. Sankoff, editors, *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM), Montreal, Canada*, volume 1848 of *Lecture Notes in Computer Science*, Springer, New York, 2000, pp. 154–165.
18. G. Lin, Z-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. *J Comput Syst Sci*, 65(3):465–480, 2002. Special issue on computational biology.
19. G. Lin, B. Ma, and K. Zhang. Edit distance between two RNA structures. *RECOMB*, Montreal, Quebec, Canada, 2001, pp. 211–220.

20. D. Sankoff and B. Kruskal. *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

21. D. Shasha and K. Zhang. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J Comput*, 18(6):1245–1262, 1989.

22. S. Vialette. Pattern matching over 2-intervals sets. In A. Apostolico and M. Takeda, editors, *Proceedings of the 13th Annual Symposium Combinatorial Pattern Matching (CPM), Fukuoka, Japan*, volume 2373 of *Lecture Notes in Computer Science*, Springer, New York, 2002, pp. 53–63.

23. S. Vialette. On the computational complexity of 2-interval pattern matching problems. *Theor Comput Sci*, 312(2–3):223–249, 2004.

# 7

# ALGORITHMIC ISSUES IN DNA BARCODING PROBLEMS

Bhaskar DasGupta, Ming-Yang Kao, and Ion Măndoiu

## 7.1 INTRODUCTION

In the outbreak of an epidemic, possibly as a result of biological warfare, there is an urgent need to identify the pathogen or the family it belongs to as early as possible. Armed with the identity of the pathogen or its family, and with prior knowledge of how the pathogen typically is spread, decision makers efficiently can alert the general public and first responders on how best to stave-off the invasion. Recent advances in genomic technologies, including the availability of a whole genome sequence for numerous pathogens and the improved sensitivity of a second generation of microarray-based hybridization platforms, have opened the way for the development of highly reliable genomic-based pathogen detection systems. However, the development of such a detection system appropriate for use by first responders still raises several challenging design issues. In addition to portability and cost-effectiveness, widespread use of such systems requires rapid and reliable identification from minute amounts of genetic material of mutated or artificially engineered unknown pathogens. At the same time, these systems should provide comprehensive coverage of known or partially known pathogens, robustness of the detection algorithms against malicious adversaries, and built-in support for easy updates of the set of recognized pathogens.

As a motivation to study a basic version of barcoding problems of interest in this chapter, consider the following scenario. Classical approaches to pathogen detection

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

are based on sequencing and direct microarray hybridization [16,24]. Although very reliable, sequencing-based detection is practically applicable only when the number of candidate pathogens is small because it requires the ability to isolate very few pathogen-specific DNA or RNA fragments. At the same time, direct microarray hybridization does not scale well with the number of potential pathogens. Reliable detection by this method requires as much as 10–20 arrayed probes per pathogen, each 70 nucleotides long [24], thus limiting the coverage of a single microarray to at most a few thousand pathogens. To overcome some of these difficulties, one employs *rapid* and *robust* computational procedures to compute *barcodes* that produce short signatures and thereby both reduce database size and optimize cost of designing the hybridization array.

In this chapter, we survey several barcoding problems that have applications as mentioned as well as in other areas, and we survey some key algorithmic techniques used in the existing literatures for these problems. We assume that the reader is familiar with the basic concepts of exact and approximation algorithms (*e.g.*, see [6,23]), basic computational complexity classes such as polynomial time (P) and nondeterministic polynomial (NP) [10,13,20], and basic notions of molecular biology such as DNA sequences [12].

## 7.2 TEST SET PROBLEMS: A GENERAL FRAMEWORK FOR SEVERAL BARCODING PROBLEMS

One of the test set problems was on the classic list of NP-complete problems given by Garey and Johnson [10]; these problems develop naturally in many other applications. One can define a general framework for test set problems in the following manner. We are given an universe of objects, family of subsets ("tests") of the universe, and a notion of "distinguishability" of pairs of elements of the universe by a collection of these tests. Our goal is to select a subset of these tests of minimum size that distinguishes every pair of elements of the universe. To be precise, each of these problems is obtained by fixing parameters in the *general* test set problem $\mathrm{TS}^{\Gamma}(k)$ as described subsequently ($2^X$ denotes the *power set* of a set $X$).

**Definitio 7.1** *(Problem $TS^{\Gamma}(k)$ with $\Gamma \subseteq 2^{\{0,1,2\}}$ and $k$ being a positive integer)*

**Instance:** $(n, \mathcal{S})$ *where* $\mathcal{S} \subset 2^{\{0,1,2,\ldots,n-1\}}$

**Terminologies:**

- *A $k$-*test *is a union of at most $k$ sets from $\mathcal{S}$*
- *For a $\gamma \in \Gamma$ and two distinct elements $x, y \in \{0, 1, 2, \ldots, n-1\}$, a $k$-test $T$ $\gamma$-distinguishes $x$ and $y$ if $|\{x, y\} \cap T| \in \gamma$*

**Valid solutions:** *A collection $\mathcal{T}$ of $k$-tests such that*

$(\forall x, y \in \{0, 1, 2, \ldots, n-1\}\ \ \forall \gamma \in \Gamma)\ \ x \neq y$

$\Longrightarrow\ \ \exists T \in \mathcal{T}$ *where $T$ $\gamma$-distinguishes $x$ and $y$*

**Objective:** *minimize* $|\mathcal{T}|$

This framework captures several "barcoding-type" problems in a few areas in bioinformatics and biological modeling such as:

**Minimum Test Collection Problem:** This problem has applications in *diagnostic testing* [10]. Here a collection of tests distinguishes two objects if a test from the collection contains exactly one of them. In our previous formalism, this is precisely $\text{TS}^{\{1\}}(1)$.

**Condition Cover Problem:** Karp *et al.* [15] considered a problem of verifying a multioutput feedforward Boolean circuit as a model of biological pathways. This problem can be phrased like the minimum test collection problem, except that two elements are distinguished by a collection of tests if one tests contains exactly one of them, and another contains both or none of them. Assuming that the allowed perturbations are given as *part of the input*, this problem is identical to $\text{TS}^{\{1\},\{0,2\}}(1)$.

**String Barcoding Problem** ($\text{SB}^{\Sigma}(k)$)**:** In the "basic" version of this problem corresponding to $k = 1$, first discussed by Rash and Gusfield [21], the universe $U$ consists of sequences (strings) over an alphabet $\Sigma$, and any string $v \in \Sigma^*$ defines a test $T_v$ consisting of a collection of strings from $U$ in which $v$ appears[1]. Because this chapter is significantly concerned with this basic version, we write the problem definition explicitly for the convenience of the reader. We are given a set $\mathcal{S}$ of sequences over some alphabet $\Sigma$. For a *fixed* set of $m$ "distinguisher" sequences $\vec{\mathbf{t}} = (\mathbf{t}_0, \ldots, \mathbf{t}_{m-1})$, the *barcode* code$(s, \vec{\mathbf{t}})$ for each $s \in S$ is defined to be the Boolean vector $(c_0, c_1, c_{m-1})$ where $c_i$ is 1 if $t_i$ is a substring of $s$. We say that the set of distinguishers $\vec{\mathbf{t}}$ defines a *valid* barcode if for any two distinct strings $s, s' \in \mathcal{S}$, code$(s, \vec{\mathbf{t}})$ is different from code$(s', \vec{\mathbf{t}})$. Then the basic version $\text{SB}^{\Sigma}(1)$ is defined as follows:

**Instance:** $\mathcal{S} \subset \Sigma^*$
**Valid solutions:** a set of distinguisher sequences $\vec{\mathbf{t}}$ defining a valid barcode
**Objective:** minimize $|\vec{\mathbf{t}}|$

As an example, let $\Sigma = \{\text{A, C, T, G}\}$ and $\mathcal{S} = \{$AAC, ACC, GGGG, GTGTGG, TTTT$\}$. Then, the set of four distinguishers $\vec{\mathbf{t}} = \{$A, CC, TTT, GT$\}$ defines the set of valid barcodes for the input sequences in $\mathcal{S}$ as follows:

| | A | CC | TTT | GT |
|---|---|---|---|---|
| AAC | 1 | 0 | 0 | 0 |
| ACC | 1 | 1 | 0 | 0 |
| GGGG | 0 | 0 | 0 | 0 |
| GTGTGG | 0 | 0 | 0 | 1 |
| TTTT | 0 | 0 | 1 | 0 |

[1] $\Sigma^*$ is the standard notation of denoting the set of all possible strings formed by a concatenation of zero or more symbols of $\Sigma$.

The name "string barcoding" derives from the fact that the Boolean vector indicating the occurrence (as a substring) of the tests from an arbitrary collection of tests in a given input sequence is referred to as the "barcode" of the given sequence with respect to this collection of tests. Motivations for investigating these problems come from several sources, such as

- Database compression and fast database search for DNA sequences
- DNA microarray designs for efficient virus identification in which the immobilized DNA sequences at an array element are from a set of barcodes

In general, for $k > 1$, a test can be defined by a set $T$ of at most $k$ strings and $u \in U$ passes test $T$ is one of the strings in $T$ that is a substring of $u$; such tests may be feasible in practice as the one-string tests.

**Minimum Cost Probe Set Problem with a Threshold** $r$ **($\mathrm{MCP}^{\Sigma}(r)$):** This problem is very similar to string barcoding and was considered first by Borneman *et al.* [3]. Denote by $\mathrm{oc}(x, y)$ the number of occurrences of $x$ in $y$ as a substring, For a fixed set of $m$ distinguisher sequences $\vec{\mathbf{t}} = (t_0, t_1, \ldots, t_{m-1})$, an $r$-barcode $\mathrm{code}(s, \vec{\mathbf{t}})$ for any sequence $s$ is defined to be the vector $(c_0, c_1, \ldots, c_{m-1})$ where $c_i = \min\{r, \mathrm{oc}(t_i, s)\}$. Given a set $\mathcal{S}$ of sequences over some alphabet $\Sigma$, $\vec{\mathbf{t}}$ defines a *valid* $r$-barcode if for any two distinct strings $s, s' \in \mathcal{S}$, $\mathrm{code}(s, \vec{\mathbf{t}})$ is different from $\mathrm{code}(s', \vec{\mathbf{t}})$. $\mathrm{MCP}^{\Sigma}(r)$ now is defined as follows:

**Instance:** $(r, \mathcal{S}, \mathcal{P})$ where $\mathcal{S}, \mathcal{P} \subset \Sigma^*$
**Valid solutions:** a set of distinguisher sequences $\vec{\mathbf{t}} \subseteq \mathcal{P}$ defining a valid $r$-barcode
**Objective:** minimize $|\vec{\mathbf{t}}|$

This problem was used in [3] for minimizing the number of oligonucleotide probes needed for analyzing populations of ribosomal RNA gene (rDNA) clones by hybridization experiments on DNA microarrays; the probes are selected from a prespecified set $\mathcal{P}$. However, it also can be used in the context of other string barcoding approaches in which the barcodes are *integer-valued* as opposed to being Boolean.

## 7.3 A SYNOPSIS OF BIOLOGICAL APPLICATIONS OF BARCODING

Applications of barcoding techniques range from rapid pathogen identification in epidemic outbreaks to point-of-care medical diagnosis to monitoring of microbial communities in environmental studies (*e.g.*, see [3, 21]). For example, genomic-based identification of microorganisms such as viruses or bacteria is performed by spotting or synthesizing on a microarray the Watson–Crick complements of the distinguisher strings and then hybridizing to the array the fluorescently labeled DNA extracted from the unknown microorganism. Under the assumption of perfect hybridization stringency, the hybridization pattern can be viewed as a string of $k$ zeros and ones, referred to as the *barcode* of the microorganism. By construction, the barcodes corresponding to the $n$ microorganisms are distinct, and thus,

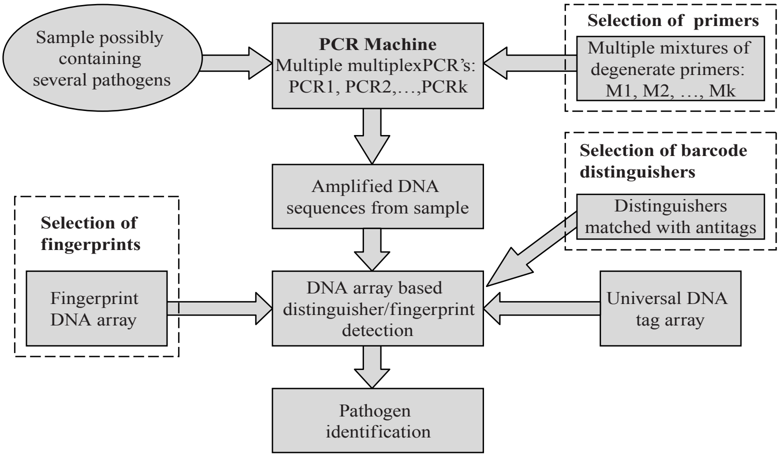

Sample possibly containing several pathogens
PCR Machine
Multiple multiplexPCR's: PCR1, PCR2,…,PCRk
Selection of primers
Multiple mixtures of degenerate primers: M1, M2, …, Mk
Amplified DNA sequences from sample
Selection of barcode distinguishers
Distinguishers matched with antitags
Selection of fingerprints
Fingerprint DNA array
DNA array based distinguisher/fingerprint detection
Universal DNA tag array
Pathogen identification

**Figure 7.1** A hypothetical architecture of FRPDS.

the barcode uniquely identifies any one of them. To improve identification robustness, one also may require *redundant distinguishability* (*i.e.*, at least $m$ different distinguishers for every pair of microorganisms, where $m > 1$ is some fixed constant) and impose a lower bound on the edit distance between any pair of selected distinguishers [21].

A hypothetical system implementing a high level architecture that meets the design criteria for a first responder pathogen detection system (FRPDS) using string barcoding is shown schematically in Figure 7.1. Such a hypothetical system includes the following three major components:

1. A component that provides *rapid amplification* of the collected genetic material (*e.g.*, degenerate oligonucleotide primer-based multiplex polymerase chain reaction (PCR)).
2. A pathogen fingerprinting and/or barcoding component (say, built around universal DNA tag arrays).
3. *Rapid* and *robust* computational procedures to compute *barcodes* that produce short signatures and thereby both reduce database size and optimize cost of designing the hybridization array.

## 7.4 SURVEY OF ALGORITHMIC TECHNIQUES ON BARCODING

In this section, we survey several algorithmic methods used to solve the barcoding problems. We then will discuss in more detail in the next two sections the set-covering and information content algorithmic approach.

### 7.4.1 Integer Programming

In [21], Rash and Gusfield discussed some experimental results for $\mathrm{SB}^{\Sigma}(1)$ but left open the exact complexity and approximability of this problem. Their algorithmic approach is based on writing the problem as an integer program and then solving it directly. Unfortunately, the run time of this approach does not scale well with the number of microorganisms and the length of the genomic sequences (*e.g.*, the largest instance sizes reported in [21] have a total genomic sequence length of around 100, 000 bases). We will not discuss the integer programming formulation in more detail because we will discuss heuristics based on set-covering methods in more detail subsequently, and an integer programming formulation for set-covering problems is well-known (*e.g.*, see [23]).

### 7.4.2 Lagrangian Relaxation and Simulated Annealing

Borneman *et al.* [3] noted that the $\mathrm{MCP}^{\Sigma}(r)$ problem was NP-complete assuming that the lengths of the sequences in the prespecified set were unrestricted and discussed some experimental results for a few heuristics that they implemented. Their algorithmic approach is based on a Lagrangian relaxation of the integer programming formulation of set cover and simulated annealing approach.

### 7.4.3 Provably Asymptotically Optimal Results

In [2] Berman *et al.* were able to provide tight theoretical worst-case approximability bounds for almost all of these problems. A summary of the results in [2] is as follows (where $\ell$ is the maximum length of any sequence in $\mathcal{S}$, $L$ is the total length of all sequences in $\mathcal{S}$, and $\varepsilon$ and $\delta$ are constants):

- $\mathrm{TS}^{\{1\}}(1)$ can be approximated to within a ratio of $1 + \ln n$ in $O(n^2|\mathcal{S}|)$ time and cannot be approximated to within a ratio of $(1 - \varepsilon)\ln n$ assuming NP $\neq$ DTIME($n^{\log\log n}$).
- $\mathrm{TS}^{\{1\},\{0,2\}}(1)$ can be approximated to within a ratio of $1 + \ln 2 + \ln n$ in $O(n^2|\mathcal{S}|)$ time and cannot be approximated to within a ratio of $(1 - \varepsilon)\ln n$ assuming NP $\neq$ DTIME($n^{\log\log n}$).
- $\mathrm{SB}^{\Sigma}(1)$ can be approximated to within a ratio of $1 + \ln n$ in $O(n^3\ell^2)$ time and cannot be approximated to within a ratio of $(1 - \varepsilon)\ln n$ assuming NP $\neq$ DTIME($n^{\log\log n}$).
- $\mathrm{MCP}^{\Sigma}(r)$ can be approximated to within a ratio of $[1 + o(1)]\ln n$ in $O(n^2|\mathcal{P}| + L\mathcal{P}|)$ time and cannot be approximated to within a ratio of $(1 - \varepsilon)\ln n$ assuming NP $\neq$ DTIME($n^{\log\log n}$).
- $\mathrm{TS}^{\{1\}}(n^{\delta})$ cannot be approximated to within a ratio of $n^{\varepsilon}$ assuming NP $\neq$ co-randomized polynomial time (RP) for any $0 < \varepsilon < \delta < 1$.
- $\mathrm{SB}^{\Sigma}(n^{\delta})$ cannot be approximated to within a ratio of $n^{\varepsilon}$ assuming NP $\neq$ co-RP for any $0 < \varepsilon < \delta < \frac{1}{2}$.

The provably optimal algorithmic approach in [2] uses an entropy-based algorithmic approach that they term as the "information content" approach. Informally, this is a greedy technique based on the information content of a partial solution; the notion of information content is related directly to the Shannon information complexity [1,22]. The greedy approach seeks to select an augmenting step for a partial solutions that maximizes the *new* information content of the augmented partial solution as compared with the partial solution. A key nontrivial step for applying this technique is to define a suitable easy-to-compute measure of the information content of a partial solution such that the monotonicity of this measure is ensured with respect to any subset of an optimal solution. The next section defines the approach more precisely.

## 7.5 INFORMATION CONTENT APPROACH

In this section, we discuss the information content approach for $\text{TS}^{\{1\}}$ as designed in [2] running in time $O(n^2|\mathcal{S}|)$ with an approximation ratio of $1 + \ln n$. Notice that the upper bound almost matches the lower bound stated in Section 7.4.3 for $\text{SB}^{\{0,1\}}$, a special case of $\text{TS}^{\{1\}}$.

For simplicity, we illustrate the approach for the problem $\text{TS}^{\{1\}}$. In the following definition and throughout the rest of this section we use $\mathcal{T} + T$ to denote $\mathcal{T} \cup \{T\}$.

**Definitio 7.2** *A set of tests $\mathcal{T} \subseteq \mathcal{S}$ defines the following:*

- *An equivalence relation $\overset{\mathcal{T}}{\equiv}$ on $\{0, 1, 2, \ldots, n-1\}$ given by $i \overset{\mathcal{T}}{\equiv} j$ if and only if $\forall T \in \mathcal{T} (i \in T \equiv j \in T)$*
- *A set of permutations $\Pi_{\mathcal{T}} = \{\pi \in$ (permutations of $\{0, 1, 2, \ldots, n-1\}$): $\forall i \in [0, n-1]\ i \overset{\mathcal{T}}{\equiv} \pi(i)\}$*
- *entropy $H_{\mathcal{T}} = \log_2 |\Pi_{\mathcal{T}}|$.*
- *information content of a $T \in \mathcal{S}$ with respect to $\mathcal{T}$, $IC(T, \mathcal{T}) = H_{\mathcal{T}} - H_{\mathcal{T}+T} = \log_2 \frac{|\Pi_{\mathcal{T}}|}{|\Pi_{\mathcal{T}+T}|}$.*

As an example, consider $\mathcal{T} = \{\{1, 2, 3, 4\}, \{1, 5, 6\}\}$ with $n = 8$. Then, the equivalence classes of $\overset{\mathcal{T}}{\equiv}$ are $\{1\}, \{2, 3, 4\}, \{5, 6\}, \{7, 8\}$, and $H_{\mathcal{T}} = \log_2((3!)(2!)(2!)) \approx 4.585$. This definition of entropy is somewhat similar (but not the same) to the one suggested in [18]. Suppose that the equivalence relation $\overset{\mathcal{T}}{\equiv}$ on $\{0, 1, 2, \ldots, n-1\}$ produces $q$ equivalence classes of size $s_1, s_2, \ldots, s_q$. Then, the entropy suggested in [18] is $\frac{1}{n} \log_2(\Pi_{i=1}^{q} s_i^{s_i})$, whereas our entropy $H_{\mathcal{T}}$ is $\log_2(\Pi_{i=1}^{q} s_i!)$.

The *information content heuristic* (ICH) is the following simple greedy heuristic:

```
$\mathcal{T} = \emptyset$
while $H_{\mathcal{T}} \neq 0$ do
    select a $T \in \mathcal{S} - \mathcal{T}$ that maximizes $IC(T, \mathcal{T})$
    $\mathcal{T} = \mathcal{T} + T$
```

The correctness of ICH follows from the fact that $H_{\mathcal{T}} = 0$ implies that the equivalence classes of $\overset{\mathcal{T}}{\equiv}$ are $n$ singleton sets $\{0\}, \{1\}, \ldots, \{n-1\}$ and the fact that if $H_{\mathcal{T}} \neq 0$, then a $T \in \mathcal{S} \setminus \mathcal{T}$ exists with $IC(T, \mathcal{T}) > 0$ (otherwise, the problem instance has no feasible solution).

To implement ICH, one iteratively maintains the equivalence classes of $\overset{\mathcal{T}}{\equiv}$ as sorted lists. We also precompute and store $\log_2(i!)$ for each $i \in [1, n]$. Given a specific $T \in \mathcal{S} - \mathcal{T}$, it is easy to compute in $O(n)$ time the equivalence classes of $\overset{\mathcal{T}+T}{\equiv}$ from the equivalence classes of $\overset{\mathcal{T}}{\equiv}$ because an equivalence class $E$ of $\overset{\mathcal{T}}{\equiv}$ is either an equivalence class of $\overset{\mathcal{T}+T}{\equiv}$, or it is partitioned into two equivalence classes $E_1 = E \cap T$ and $E_2 = E - E_1$ of $\overset{\mathcal{T}+T}{\equiv}$; the first case contributes nothing to $IC(T, \mathcal{T})$, whereas the second case adds $\log_2 \binom{|E|}{|E_1|}$ to $IC(T, \mathcal{T})$.

The performance guarantee of this approach is given by the following theorem proved in [2] using a very careful amortized analysis.

**Theorem 7.1** *[2] The previous approach yields:*

- *for* $TS^{\{\{1\}\}}$ *an approximation ratio of* $1 + \ln n$*;*
- *for* $TS^{\{\{1\},]\{0,2\}\}}$ *an approximation ratio of* $1 + \ln 2 + \ln n$*;*
- *for* $MCP^{\Sigma}(r)$ *an approximation ratio of* $1 + \ln n + \ln \log_2(r' + 1)$*, where* $r' = \min\{r, n\}$.

## 7.6 SET-COVERING APPROACH

Methods based on this approach enable distinguisher selection based on *whole genomic sequences* of hundreds of microorganisms of up to bacterial size on a well-equipped workstation and can be parallelized easily to extend further the applicability range to thousands of bacterial size genomes. Whole-genome-based selection is beneficial in at least two significant ways. First, it simplifies assay design because the DNA of the unknown pathogen can be amplified using inexpensive general-purpose whole-genome amplification methods such as specialized forms of degenerate primer multiplex PCR [4] or multiple displacement amplification [9]. Second, whole-genome-based selection results in a reduced number of distinguishers, often very close to the information theoretic lower bound of $\lceil \log_2 n \rceil$.

Set-covering approaches are based on a simple greedy selection strategy; in every iteration we pick a substring that distinguishes the largest number of not-yet-distinguished pairs of genomic sequences. This selection strategy is an embodiment of the greedy set-cover algorithm (*e.g.*, see [23]) for a problem instance with $O(n^2)$ elements corresponding to the pairs of sequences. Hence, by a classical result of [5, 14, 17], the algorithm guarantees an approximation factor of $2 \ln n$ for the barcoding problem. Experimental results provided in [7, 8] show that our set-cover greedy algorithm produces solutions of virtually identical quality to those obtained by the information content heuristic.

The set-cover greedy algorithm is extremely versatile and can be extended easily to handle redundancy and minimum edit distance constraints as well as other biochemical constraints on individual distinguisher sequences. Furthermore, the greedy set-cover algorithm also can take into account genomic sequence uncertainties expressed in the form of degenerate bases. Although degenerate bases are ubiquitous in genomic databases, previous works have not recognized the need to handle them properly.

### 7.6.1 Set-Covering Implementation in More Detail

In this section, for simplicity, we present the implementation of the set-cover greedy algorithm as provided in [7, 8] in the context of the basic version of the string barcoding problem only. Implementation modifications needed to handle the robust barcoding problem in its full generality are available in [8].

The implementation of the set-cover greedy algorithm has two main phases: a *candidate generation phase* and a *candidate selection phase*.

In the candidate generation phase, a representative set of candidate distinguishers is generated from the given genomic sequences. Essentially, they use an incremental algorithm for quickly generating a representative set of candidate distinguishers and collecting all their occurrences in the given genomic sequences. For each generated candidate, we also compute the list of sequences with which the candidate has perfect matches; this information is needed in the candidate selection phase. To reduce the number of candidates, we avoid generating any substring that appears in all genomic sequences, which typically eliminates very short candidates. For each genomic sequence, we also make sure to generate only one of the substrings that appear exclusively in that sequence; this optimization eliminates from consideration most candidate distinguishers above a certain length. Unlike the suffix tree method proposed by Rash and Gusfield [21], this approach may generate multiple candidates that appear in the same set of $k$ genomic sequences (for $1 < k < n$). However, the penalty of having to evaluate redundant candidates in the candidate selection phase is offset in practice by the faster candidate generation time. Efficient implementation of candidate elimination rules is achieved by generating candidates in increasing order of length and using exact match positions for candidates of length $l - 1$ when generating candidates of length $l$. For each position $p$ in the input genomic sequences, we also maintain a flag to indicate whether the algorithm should evaluate candidate substrings starting at $p$. The possible values for the flag are TRUE (the substring of current length starting at $p$ is a possible candidate), FALSE (we already have saved the substring of current length starting at $p$ as a candidate), or DONE (all candidates containing as prefix the substring of current length starting at $p$ are redundant, *i.e.*, the position can be skipped for all remaining candidate lengths). Initially, all flags are set to TRUE. The FALSE flags are reset to TRUE whenever we increment the candidate length; however, we never reset DONE flags.

For every candidate length $l$, candidate evaluation proceeds sequentially over all positions of the genomic sequences. Whenever we reach a position $p$ whose flag is set to TRUE, we use the list of matches for the substring of length $l - 1$ starting at $p$

```
Input: Set $C$ of candidate distinguishers
Output: Set $D$ of selected distinguishers

$D \leftarrow \emptyset$; For every $c \in C$, $\Delta_{\text{old}}(c) \leftarrow \infty$
Repeat
    $\Delta^* \leftarrow 0$
    For every $c \in C$ with $\Delta_{\text{old}}(c) > \Delta^*$ do    // Because $\Delta(c, D) \leq \Delta_{\text{old}}(c)$, $c$ can be
            ignored if $\Delta_{\text{old}}(c) \leq \Delta^*$
        $\Delta_{\text{old}}(c) \leftarrow \Delta(c, D)$
        If $\Delta(c, D) > \Delta^*$ then $\Delta^* \leftarrow \Delta(c, D)$; $c^* \leftarrow c$
    If $\Delta^* > 0$ then $D \leftarrow D \cup \{c^*\}$
While $\Delta^* > 0$
Return $D$
```

**Figure 7.2** The greedy candidate selection algorithm.

(or a linear time string matching algorithm if $l$ is the minimum candidate length) to determine the list of matches for the substring of length $l$ starting at $p$, and set the flag to FALSE for all positions in which these matches occur. If the substring of length $l$ starting at $p$ has matches only within the source sequence, and we have already generated a "unique" candidate for this sequence, then we discard the candidate and set the flag of $p$ to DONE.

A further speed-up technique is to generate candidate distinguishers from a strict subset of the input sequences. Although this speedup potentially can affect solution quality, experimental results show that the solution quality loss for whole-genome barcoding is minimal, even when we generate candidates based on a single input sequence, which corresponds to preassigning a barcode of all 1's to this sequence.

After the set of candidates is generated, we select the final set of distinguishers in the greedy phase of the algorithm (Figure 7.2). We start with an empty set of distinguishers $D$. Although pairs of sequences are not yet distinguished by $D$, we loop over all candidates and compute for each candidate $c$ the number $\Delta(c, D)$ of pairs of sequences that are distinguished by $c$ but not by $D$, then add the candidate $c$ with the largest $\Delta$ value to $D$.

Two sequences $s$ and $s'$ are distinguished by a candidate $c$ if and only if exactly one of $s$ and $s'$ appears in the list $P_c$ of perfect matches of $c$, which is available from the candidate generation phase. A simple method for computing $\Delta$ values is to maintain an $n \times n$ symmetric matrix indicating which of the pairs of sequences already are distinguished and then to probe the $|P_c| \times (n - |P_c|)$ entries in this matrix corresponding to pairs $(s, s')$ with $s \in P_c$ and $s' \notin P_c$ when computing $\Delta(c, D)$. A more efficient method is based on maintaining the partition defined on the set of sequences by $D$. If the partition defined by $D$ consists of sets $S_1, \ldots, S_k$, then we can compute $\Delta(c, D)$ in $O(k + |P_c|) = O(n)$ time using the observation that

$$\Delta(c, D) = \sum_{i=1}^{k} |S_i \cap P_c| \times |S_i \setminus P_c| \tag{7.1}$$

In addition to the fast partition-based computation, the implementation of the greedy selection phase uses a lazy strategy for updating the $\Delta$ values, based on the observation that they are monotonically nonincreasing during the algorithm (see Figure 7.2). Thus, the efficient implementation of the greedy selection phase of the algorithm combines a partition-based method for computing the coverage gain of candidate distinguishers (this method first was proposed in the context of the information content heuristic in [2]) with a "lazy" strategy for updating coverage gains.

## 7.7 EXPERIMENTAL RESULTS AND SOFTWARE AVAILABILITY

The authors in [7, 8] performed experiments on both randomly generated instances and whole microbial genomes extracted from the National Center for Biotechnology Information (NCBI) databases. Random test cases were generated from the uniform distribution induced by assigning equal probabilities to each of the four nucleotide; these test cases do not contain any nucleotides with degeneracy greater than 1. The NCBI test case represents a selection of 29 complete microbial sequences, varying in length between 490, 000 and 4, 750, 000 bases (more than 76 million bases in total). All experiments were run on a PowerEdge 2600 Linux server with 4 Gb of RAM and dual 2.8 GHz Intel Xeon central processing units (CPUs)—only one of which is used by the sequential algorithms.

### 7.7.1 Randomly Generated Instances

As described in Section 7.6.1, there are two main phases in the algorithm: candidate distinguisher generation and greedy candidate selection. Results were reported about the average candidate selection CPU time for $n$ random sequences of length 10,000 and redundancy 1, averaged over 10 instances of each size. Combining the two speed-up techniques for this phase (partition-based coverage gain computation and lazy update of candidate gains) results in more than two orders of magnitude reductions in run time.

A further speedup technique is to generate candidate distinguishers from a select subset of the input sequences. Although this speed-up potentially can affect solution quality, the results showed that on large instances, the solution quality loss is minimal even when we generate candidates based on a single input sequence; this corresponds to preassigning a barcode of all 1's to this sequence. The technique reduces significantly both the memory requirement (which is proportional to the number of candidates and the number of times they match input sequences) and the run time required for candidate generation and greedy selection.

The quality of the solution in the simulations was as follows. The number of distinguishers returned by the set-cover greedy algorithm were reported for redundancy varying between 1 and 20 on between 10 and 1,000 random sequences of length 10,000. These results were compared with the results obtained by the information content heuristic results of [2] as well as the information theoretic lower bound of $\lceil \log_2 n \rceil$ for when the redundancy requirement is 1. The number of distinguishers returned by the set-cover greedy algorithm was virtually identical to that returned by the information content heuristic, despite the latter one having a better approximation guarantee. Furthermore, the results for redundancy were within 50% of the information theoretic lower bound for the range of instance sizes considered in this experiment. The gap between the solutions returned by the algorithms and the lower bound does increase with the number of sequences; however, it is not clear how much of this increase is caused by degrading algorithm solution quality and how much is caused by degrading lower bound quality.

### 7.7.2 Real Data

The algorithm was run on a set of 29 complete microbial genomic sequences extracted from NCBI databases [19]. Sequence lengths in the set vary between 490,000 bases and 4.75 million bases, with an average length of 2.6 million bases (more than 76 million bases total). In these experiments, we varied the redundancy requirement, from 1 to 20. To see the effect of length and edit distance requirements on the number of distinguishers, for each redundancy requirement, they computed both an unconstrained solution and a solution in which distinguishers must have a length between 15 and 40, and there should be a minimum edit distance of six between every two selected distinguishers (these values are similar to those used in [21]). In all experiments, they generated candidates based only on the shortest sequence of 490,000 bases.

Naturally, meeting higher redundancy constraints requires more distinguishers to be selected. Additional length and edit distance constraints further increase the number of distinguishers, but the latter is still within reasonable limits. The length constraints reduce the number of candidates (from 1,775,471 to 122,478), which for low redundancy values, has the effect of reducing greedy selection time. However, for high-redundancy requirements, the reduction in number of candidates is offset by the increase in solution size, and greedy selection becomes more time consuming with length and edit distance than without (selection time grows roughly linearly with solution size).

### 7.7.3 Software Availability

The implementation of the set-covering approach, which was named DNA-BAR, can be used online through the web interface provided at http://dna.engr.uconn.edu/software/DNA-BAR/. The open source C code, released under the GNU General Public License, is also available at this address.

## 7.8 CONCLUDING REMARKS

In many practical pathogen identification applications, collected biological samples may contain the DNA of multiple pathogens. This issue is considered to be particularly significant in medical diagnosis applications (*e.g.*, see [11] for studies in detecting more than one human papilomavirus (HPV) genotype with a varying rate of multiple HPV infections carried by the same HPV carrier). A significant future research direction could be to develop extensions of the barcoding technique that reliably can detect multiple pathogens for a given bound on the number of pathogens present.

## ACKNOWLEDGMENTS

Bhaskar DasGupta was supported in part by NSF Grants IIS-0346973, IIS-0612044 and DBI-0543365. Ion Măndoiu was supported in part by NSF Grant DBI-0543365. The authors also would like to thank all their collaborators in these research topics.

## REFERENCES

1. Y.S. Abu-Mostafa, editor. *Complexity in Information Theory*. Springer Verlag, New York, 1986.
2. P. Berman, B. DasGupta, and M.-Y. Kao. Tight approximability results for test set problems in bioinformatics. *J Comput Syst Sci*, 71(2):145–162, 2005.
3. J. Borneman, M. Chrobak, G.D. Vedova, A. Figueora, and T. Jiang. Probe selection algorithms with applications in the analysis of microbial communities. *Bioinformatics*, 1:1–9, 2001.
4. V.G. Cheung and S.F. Nelson. Whole genome amplification using a degenerate oligonucleotide primer allows hundreds of genotypes to be performed on less than one nanogram of genomic dna. *PNAS*, 93:14676–14679, 1996.
5. V. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.
6. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2001.
7. B. DasGupta, K. Konwar, I. Mandoiu, and A. Shvartsman. DNA-BAR: Distinguisher Selection for DNA barcoding. *Bioinformatics*, 21(16):3424–3426, 2005.
8. B. DasGupta, K. Konwar, I. Mandoiu, and A. Shvartsman. Highly scalable algorithms for robust string barcoding. *Int J Bioinformatics Res Appl*, 1(2):145–161, 2005.
9. F.B. Dean, S. Hosono, *et al*. Comprehensive human genome amplification using multiple displacement amplification. *PNAS*, 99:5261–5266, 2002.
10. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman, New York, 1979.
11. B. Gharizadeh, M. Käller, P. Nyrén, A. Andersson, M. Uhlén, J. Lundeberg, and A. Ahmadian. Viral and microbial genotyping by a combination of multiplex competitive hybridization and specific extension followed by hybridization to generic tag arrays. *Nucleic Acids Res*, 31(22), 2003.
12. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, UK, 1997.
13. D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishers, Boston, MA, 1996.

14. D.S. Johnson. Approximation algorithms for combinatorial problems. *J Comput Syst Sci*, 9:256–278, 1974.
15. R.M. Karp, R. Stoughton, and K.Y. Yeung. Algorithms for choosing differential gene expression experiments. *Proceedings of Lyon, France, Third Annual International Conference on Computational Molecular Biology*, 1999, pp. 208–217.
16. T.G. Ksiazek, D. Erdman, *et al*. A novel coronavirus associated with severe acute respiratory syndrome. *New Engl J Med*, 348:1953–1966, 2003.
17. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math*, 13, 383–390, 1975.
18. B.M.E. Moret and H.D. Shapiro. On minimizing a set of tests. *SIAM J Sci Stat Comput*, 6:983–1003, 1985.
19. NCBI Completed Microbial Genomes. http://www.ncbi.nlm.nih.gov/genomes/microbes/complete.html.
20. C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.
21. S. Rash and D. Gusfield. String Barcoding: Uncovering Optimal Virus Signatures. *Sixth Annual International Conference on Computational Biology*, Washington D.C., USA, 2002, pp. 54–261.
22. C.E. Shannon. Mathematical theory of communication. *Bell Syst Tech J*, 27:379–423, 1948.
23. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, New York, 2001.
24. D. Wang, L. Coscoy, *et al*. Microarray-based detection and genotyping of viral pathogens. *PNAS*, 99:15687–15692, 2002.

# 8

# RECENT ADVANCES IN WEIGHTED DNA SEQUENCES

Manolis Christodoulakis and Costas S. Iliopoulos

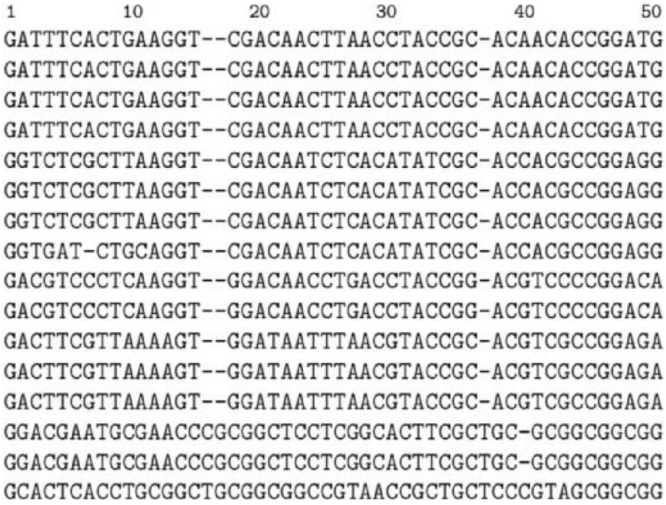
## 8.1 INTRODUCTION

It is a well-known fact, referred to as "the first fact of biological sequence analysis" [21], that biological sequences that are similar to each other tend to have similar two- or three-dimensional structure and/or perform similar biological functions. This fact indeed is used to infer the function of a given gene or protein by finding similar sequences whose functionality already is known [35].

One of the most fundamental tools for visualising similarity between two sequences is the *string alignment*. Numerous pairwise alignment methods exist, including the *dot matrix analysis* [19], various forms of *dynamic programming*—(*e.g.*, the local alignment Smith–Waterman algorithm [40], and the Needleman–Wunsch global alignment algorithm [34]) as well as *heuristic methods* (*e.g.*, FASTA [30, 36] and BLAST [2]).

Although the importance of pairwise sequence alignment cannot be overstated, it seems that aligning more than two sequences concurrently can be even more helpful in identifying similarities. Subsequences that are conserved among all (or, most) sequences and, therefore, possibly characterize all sequences at hand are easier to identify. Similar to pairwise alignments, several types of multiple alignments also exist, like global alignments (*e.g.*, ClustalW [42]) and local alignments (*e.g.*, Dialign [33]). Figure 8.1, shows a small portion of the alignment of a set of *serine/arginine-rich*

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya



```
1       10        20        30        40        50
GATTTCACTGAAGGT--CGACAACTTAACCTACCGC-ACAACACCGGATG
GATTTCACTGAAGGT--CGACAACTTAACCTACCGC-ACAACACCGGATG
GATTTCACTGAAGGT--CGACAACTTAACCTACCGC-ACAACACCGGATG
GATTTCACTGAAGGT--CGACAACTTAACCTACCGC-ACAACACCGGATG
GGTCTCGCTTAAGGT--CGACAATCTCACATATCGC-ACCACGCCGGAGG
GGTCTCGCTTAAGGT--CGACAATCTCACATATCGC-ACCACGCCGGAGG
GGTCTCGCTTAAGGT--CGACAATCTCACATATCGC-ACCACGCCGGAGG
GGTGAT-CTGCAGGT--CGACAATCTCACATATCGC-ACCACGCCGGAGG
GACGTCCCTCAAGGT--GGACAACCTGACCTACCGG-ACGTCCCCGGACA
GACGTCCCTCAAGGT--GGACAACCTGACCTACCGG-ACGTCCCCGGACA
GACTTCGTTAAAAGT--GGATAATTTAACGTACCGC-ACGTCGCCGGAGA
GACTTCGTTAAAAGT--GGATAATTTAACGTACCGC-ACGTCGCCGGAGA
GACTTCGTTAAAAGT--GGATAATTTAACGTACCGC-ACGTCGCCGGAGA
GGACGAATGCGAACCCGCGGCTCCTCGGCACTTCGCTGC-GCGGCGGCGG
GGACGAATGCGAACCCGCGGCTCCTCGGCACTTCGCTGC-GCGGCGGCGG
GCACTCACCTGCGGCTGCGGCGGCCGTAACCGCTGCTCCCGTAGCGGCGG
```

**Figure 8.1** Part of a multiple alignment of the SC35 protein across species and alleles.

(SR) proteins SC35 across species and alleles, obtained by the implementation of ClustalW available on the web [17].

Multiple alignments capture the essence of a family of related sequences, but as such families grow larger, it is not easy to manipulate the whole alignment; one needs some sort of *representation* for the family. Any such representation ought to be easier to manipulate but still capable of revealing the conserved subsequences of the sequences in the family. Three main types of representations exist for families of aligned sequences, each with its own advantages and disadvantages.

The simplest form of representation is the *consensus string*, which is a string in which each position records the nucleotide (or amino acid) that occurs more frequently in that position of the multiple alignment. The simplicity in this model is obvious, but it lacks recording important information, such as the variation in the positions.

For this reason, *regular expressions* were introduced. In their least complex form, regular expressions are sequences in which a position contains either a single symbol if that symbol is conserved among the family or the set of all symbols that occur at that position within the family. For instance, for the multiple alignment shown in Figure 8.1 the regular expression would look like

```
G[ACG][ACT][CGT][AGT][ACT][-ACG]...
```

Regular expressions, or variations of them, are used widely in biological databases; PROSITE [7, 8] for example, is a database that uses a more sophisticated form of regular expression (called *patterns*), which allows for variable-size gaps, *etc.* Regular expressions do record the various symbols occurring at each position of

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| A | 0 | 0.56 | 0.19 | 0 | 0.07 | |
| C | 0 | 0.06 | 0.31 | 0.38 | 0 | |
| G | 1.00 | 0.38 | 0 | 0.19 | 0.13 | ··· |
| T | 0 | 0 | 0.5 | 0.43 | 0.80 | |
| - | 0 | 0 | 0 | 0 | 0 | |

**Figure 8.2** Weighted sequence.

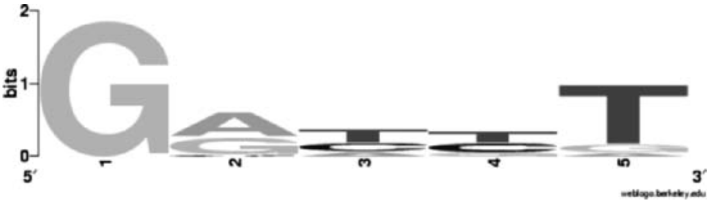


**Figure 8.3** Sequence logo: a graphical representation of a weighted sequence.

the alignment, but they do not store any information regarding how conserved each symbol is at each position.

*Weighted sequences*[1] [20] were introduced as a means of overcoming the limitations of consensus strings and regular expressions. A weighted sequence not only records the various symbols occurring at each position but also a score or weight associated with each symbol, which often corresponds to the relative frequency of the symbol at the particular position. Weighted sequences can be visualized as a matrix in which rows correspond to the symbols of the alphabet, columns correspond to the positions of the sequence, and in the cells, the probabilities (relative frequencies) of the symbols are stored. For example, Figure 8.2 shows the first five positions of the weighted sequence that corresponds to the alignment shown in Figure 8.1.

Alternatively, weighted sequences can be represented by *sequence logos* [39]. The latter constitute graphical representations of weighted sequences in the following manner: first, the symbols occurring at each position are drawn with size proportional to their relative frequency for that particular position, and second, the overall height of a column is drawn in proportion to the conservation at that position. Figure 8.3 shows the first five positions of the logo of our usual example, drawn with the web tool *weblogo* [16].

In this chapter, we review existing algorithms to manipulate and efficiently extract information from weighted sequences. More specifically, after formally introducing the notions used throughout the chapter (Section 8.2), we study the problems of indexing weighted sequences (Section 8.3), performing pattern matching (Section 8.4),

[1] Also known as *sequence profiles*, *position weight matrices* (PWMs), position frequency matrices (PFM), or position specific score matrices (PSSM).

locating repetitive elements (Section 8.6), and discovering motifs (Section 8.7). Finally, in Section 8.8 we conclude the chapter and describe open algorithmic problems on weighted sequences that still need to be addressed.

In all cases examined in this chapter, we assume that the main sequence, the "text," as we call it, is a weighted sequence and the pattern, or the repetitive elements or motifs that are extracted, are regular (nonweighted) strings. Algorithmic problems similar to the ones studied here, but where the pattern is itself a weighted sequence and the text is either weighted or not is also interesting both algorithmically and in terms of their applications in bioinformatics but are beyond the scope of this chapter. For a review of such algorithms, the reader may refer to [37].

## 8.2 PRELIMINARIES

### 8.2.1 Strings

An *alphabet* is a nonempty finite set of symbols $\Sigma = \{a_1, \ldots, a_\sigma\}$. The *size* of $\Sigma$, denoted by $|\Sigma|$, is the number of distinct symbols in $\Sigma$, (*i.e.*, $\sigma$). A *string*[2] $x = x[1] \ldots x[n]$ over an alphabet $\Sigma$ is a sequence of zero or more symbols $x[i] \in \Sigma$. The string that contains zero symbols is known as the *empty string* and is denoted by $\varepsilon$. The set of all strings over an alphabet $\Sigma$ is denoted by $\Sigma^*$. The *length* of a string $x = x[1] \ldots x[n]$, denoted by $|x|$, is the number of symbols in $x$ (*i.e.*, $n$).

A string $y$ is a *substring* (or a *factor*) of a string $x$ if and only if strings $u, v \in \Sigma^*$ exist such that $x = uyv$. Similarly, $y$ is said to be a *prefix* of $x$ if and only if $x = yv$, and $y$ is said to be a *suffix* of $x$ if and only if $x = uy$. $y$ is called a *subsequence* of $x$ if $y$ is obtained by deleting zero or more symbols at any positions from $x$.

Let $x = x[1] \ldots x[n]$ and $y = y[1] \ldots y[m]$; $x$ is said to *overlap* with $y$ by $i$ symbols if a suffix of $x$ equals a prefix of $y$, $x[n-i+1..n] = y[1..i]$, for $i \in 1 \cdot . \min(n, m)$.

A string $y = y[1] \ldots y[m]$ $(m > 0)$ is said to be a *repetition* in a string $x = x[1] \ldots x[n]$ if there are $r \geq 2$ distinct positions $j_1, j_2, \ldots, j_r \in 1 \ldots n - m + 1$ such that $x[j_q \ldots j_q + m - 1] = y$, for all $q \in 1 \ldots r$. Moreover, if there are $q_1, q_2, \ldots, q_s \in 1..r$, $q_1 < q_2 < \ldots < q_s$, such that $j_{q_l} = j_{q_{l-1}} + m$, for all $l \in 1..s$, then $x[j_{q_1}..j_{q_s} + m - 1]$ is called a *tandem repeat* or a *run* and can be written in short $x[j_{q_1}..j_{q_s} + m - 1] = y^s$.

A substring $y$ of a string $x$ is called a *period* of $x$ if $x = y^k y'$, where $k \geq 1$ and $y'$ is a (possibly empty) prefix of $y$.[3] The shortest period of $x$ is called *the* period of $x$.

[2] Where it is not clear from the context, we will refer to strings as *solid* (or *regular*) strings to emphasize that they are not *weighted* sequences (see Section 8.2.2).

[3] Note that the usual definition of the period of a string refers to the *length* of the string $y$, rather than $y$ itself. We prefer this definition here, however, because we will be interested in the string itself rather than its length. Although, for solid strings, the period string can be deduced from the period length, in weighted sequences this is not the case.

### 8.2.2 Weighted Sequences

A *weighted sequence* is a sequence in which each position, instead of consisting of a single symbol as in (solid) strings, it contains several symbols, each one associated with a weight. Formally, for alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\}$, a weighted sequence $w = w[1] \ldots w[n]$ over $\Sigma$ is a sequence of vectors of size $\sigma \times 1$, such that

$$w[i] = \begin{pmatrix} \pi_i(a_1) \\ \pi_i(a_2) \\ \vdots \\ \pi_i(a_\sigma) \end{pmatrix} \quad \text{for } i \in 1..n$$

where $\pi_i(a_h)$ represents the *weight* (or *score*) of the symbol $a_h$, $h \in 1..\sigma$ at position $i$ of $w$. Because weighted sequences in most cases are used to represent a set of aligned sequences, the weight associated with each symbol represents the *relative frequency* (or, in other words, the *probability*) of occurrence of the symbol at that particular position. Thus, for every position $i \in 1..n$ and every symbol $a_h \in \Sigma$,

$$\pi_i(a_h) \in 0..1 \quad \text{and} \quad \sum_{h=1}^{\sigma} \pi_i(a_h) = 1$$

The previous definition suggests that a weighted sequence can be viewed as a $\sigma \times n$ matrix, where columns represent the positions, $i \in 1 \ldots n$, of the weighted sequence, and rows represent the symbols in $\Sigma$. For this reason, a weighted sequence often is called a *position weight matrix*.[4]

For example, consider $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Then the matrix

$$w = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0.30_{\mathtt{A}} & 0.25_{\mathtt{A}} & 0.00_{\mathtt{A}} & 0.40_{\mathtt{A}} & 0.80_{\mathtt{A}} & 0.00_{\mathtt{A}} \\ 0.00_{\mathtt{C}} & 0.25_{\mathtt{C}} & 1.00_{\mathtt{C}} & 0.20_{\mathtt{C}} & 0.05_{\mathtt{C}} & 0.50_{\mathtt{C}} \\ 0.20_{\mathtt{G}} & 0.50_{\mathtt{G}} & 0.00_{\mathtt{G}} & 0.20_{\mathtt{G}} & 0.10_{\mathtt{G}} & 0.00_{\mathtt{G}} \\ 0.50_{\mathtt{T}} & 0.00_{\mathtt{T}} & 0.00_{\mathtt{T}} & 0.20_{\mathtt{T}} & 0.05_{\mathtt{T}} & 0.50_{\mathtt{T}} \end{pmatrix} \end{array} \qquad (8.1)$$

represents a weighted sequence of length 6. Note that, for clarity, next to each probability, the symbol that corresponds to that probability is written. At position 1, the probability of A is 30%, C is 0%, G is 20%, and T is 50%.

A symbol $a$ is said to *occur* or *match* at position $i$ of a weighted sequence $w = w[1] \ldots w[n]$ if and only if $\pi_i(a) > 0$. A string $y = y[1] \ldots y[m]$ *occurs* (or *matches*) at position $i$ of a weighted sequence $w = w[1] \ldots w[n]$ if and only if, for all $j \in 1 \ldots m$, $y[j]$ occurs at position $w[i + j - 1]$; that is, if and only if

[4] When the weighted sequence is sparse, containing only a few symbols per position, storing a $\sigma \times 1$ vector for each position is not practical. In such cases, each position can be represented by a set of pairs of the form $(a, \pi_i(a))$, thus storing only the symbols that have nonzero probability.

$\pi_{i+j-1}(y[j]) > 0$, for all $j \in 1 \ldots m$. We also say that $y$ *matches* $w$ at position $i$ or that $y$ is a *factor* of $w$ at position $i$. For example, $y = \mathtt{ACTA}$ occurs in the weighted sequence $w$ defined earlier, at position 2 because $\pi_2(\mathtt{A}) = 0.25 > 0$, $\pi_3(\mathtt{C}) = 1 > 0$, $\pi_4(\mathtt{T}) = 0.2 > 0$, and $\pi_5(\mathtt{A}) = 0.8 > 0$. If the string $y$ occurs at position 1 of $w$, then it is called a *prefix* of $w$, whereas if it occurs at position $n - m + 1$ (thus ending at position $n$), then it is called a *suffix*.

For every occurrence of the string $y$ in $w$, we define the *probability* (or weight/score) of that occurrence as follows:

$$\Pi_i(y) = \prod_{j=1}^{m} \pi_{i+j-1}(y[j])$$

For instance, for the weighted sequence (1.1)

$$\Pi_2(\mathtt{ACTA}) = 0.25 \times 1 \times 0.2 \times 0.8 = 0.04$$

If we interpret the $\pi_i$'s as scores, rather than probabilities, then other matching measures may become interesting. For example, we could define the matching weight as the maximum of the weights of the individual symbols, $\max_{j \in 1 \ldots m}\{\pi_{i+j-1}(y[j])\}$, or as their sum $\sum_{j \in 1..m}\{\pi_{i+j-1}(y[j])\}$.

The computation of the weights of the matches plays an important role in distinguishing good, valid, matches from random ones. For this reason, algorithms dealing with weighted sequences normally require the existence of a threshold probability $k$, and any matches with a probability less than $k$ are discarded.

## 8.3 INDEXING

The notion of *indexing* refers to preprocessing (usually large) input sequences to speed up queries that will follow, like pattern matching or finding repetitions. In this section, we present two indexing data structures, the weighted suffix tree (WST) and the property suffix tree (PST), which both index maximal solid factors of a weighted sequence. The great advantage of using such data structures is that they allow various existing algorithms on solid strings to be applied transparently (or almost transparently) on weighted sequences.

### 8.3.1 Weighted Suffix Tree

The weighted suffix tree was introduced by Iliopoulos *et al.* [22, 23] as an efficient data structure for computing different types of repetitions and regularities in weighted sequences, whereas in [29], the authors extended its use to other applications, such as pattern matching and motif extraction.

For solid strings, the *suffix tree* is one of the most fundamental data structures. Simply put, the suffix tree of a string is a compact trie of all suffixes of the string.

Weiner [45] and McCreight [32] presented off-line linear-time algorithms to construct the suffix tree, whereas more recently Ukkonen [43] devised an online linear-time algorithm. For a given string $x$, the suffix tree is constructed on $x\$$, where \$ is a unique symbol not occurring in $x$, to ensure that each suffix is associated with a (unique) leaf of the suffix tree. It is clear that the number of suffixes in a string and thus the number of leaves in the suffix tree is linear.

On the contrary, the number of (solid) suffixes in a weighted sequence can be in the worst case exponential ($|\Sigma|^n$ for a weighted sequence of length $n$), and thus, a suffix tree containing *all* the suffixes is impractical here. However, as discussed in Section 8.2.2, we hardly ever are interested in factors of a weighted sequence with very little probability of occurrence, and to accommodate this, we normally are given a cut-off probability $k$. Consequently, the weighted suffix tree need not store suffixes or factors with probability less than $k$.

Let $w = w[1] \ldots w[n]$ be a weighted sequence and let WST($w$) denote its weighted suffix tree. For every position $i \in 1..n$ of $w$, a set $\mathcal{X}_i$ is constructed that contains for every (solid) suffix that starts at position $w[i]$, the longest prefix of it that has a probability greater than or equal to $k$; let $X_{i,j}$ denote the $j$-th suffix in arbitrary ordering. A leaf $v$ of WST($w$) is labelled with index $i$ if and only if a prefix exists of one of the suffixes starting at position $i$, say $X_{i,j}$, whose probability is greater than or equal to $k$. Notice that in contrast to regular suffix trees, a leaf of the WST does not necessarily correspond to a suffix but rather to the longest prefix of the particular suffix whose probability is greater than or equal to $k$. Interestingly, in [29], it was proven that the size of $\mathcal{X}_i$ is bounded by $O(|\Sigma|^{\log k/\log(\frac{k}{k-1})})$, which is a constant (considering $\Sigma$ and $k$ as constants).

Figure 8.4 shows the weighted suffix tree of the weighted sequence

$$w = \left(\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
1_{\mathtt{A}} & 0_{\mathtt{A}} & 0_{\mathtt{A}} & 0_{\mathtt{A}} & 0.5_{\mathtt{A}} & 0_{\mathtt{A}} & 0_{\mathtt{A}} & 0.5_{\mathtt{A}} & 0_{\mathtt{A}} & 0_{\mathtt{A}} & 0_{\mathtt{A}} \\
0_{\mathtt{C}} & 1_{\mathtt{C}} & 0_{\mathtt{C}} & 0_{\mathtt{C}} & 0.5_{\mathtt{C}} & 0_{\mathtt{C}} & 1_{\mathtt{C}} & 0.3_{\mathtt{C}} & 0_{\mathtt{C}} & 0_{\mathtt{C}} & 0_{\mathtt{C}} \\
0_{\mathtt{G}} & 0_{\mathtt{G}} & 0_{\mathtt{G}} & 0_{\mathtt{G}} & 0.0_{\mathtt{G}} & 0_{\mathtt{G}} & 0_{\mathtt{G}} & 0.0_{\mathtt{G}} & 0_{\mathtt{G}} & 0_{\mathtt{G}} & 0_{\mathtt{G}} \\
0_{\mathtt{T}} & 0_{\mathtt{T}} & 1_{\mathtt{T}} & 1_{\mathtt{T}} & 0.0_{\mathtt{T}} & 1_{\mathtt{T}} & 0_{\mathtt{T}} & 0.2_{\mathtt{T}} & 1_{\mathtt{T}} & 1_{\mathtt{T}} & 1_{\mathtt{T}}
\end{array}\right)$$

with cut-off probability $k = 0.25$. Notice that, for instance, the suffix ATCCTTT, starting at position 5 is not stored in the WST because its probability is 0.15.

The weighted suffix tree can be built in linear time in the following stages [29]:

**Colouring.** The weighted sequence is scanned, and all positions are marked with a color according to the following criteria[5]:

- A position $i$ is marked as *black*, if $\pi_i(a) < 1 - k$ for all $a \in \Sigma$; these positions are called *branching* positions because more than one symbol occurring at this position may have probability $k$ or more.

[5] There is an implicit assumption that $k < 0.5$ here; when $k \geq 0.5$ the coloring process is even simpler as there will be no branching positions.

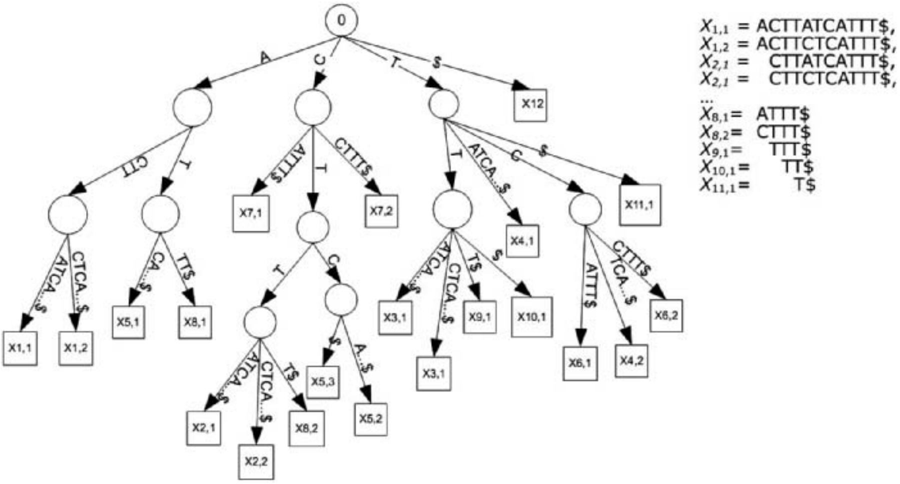


**Figure 8.4** Example of a weighted suffix tree [29].

- A position $i$ is marked as *gray*, if one $a \in \Sigma$ exists such that $\pi_i(a) \geq 1 - k$; in these positions, although several symbols occur, only one of them eventually "survives" because only one symbol can have a probability greater than or equal to $k$ and thus can contribute to a substring with a probability of at least $k$.
- A position $i$ is marked as *white*, if there exists one $a \in \Sigma$ such that $\pi_i(a) = 1$; these are called *solid* positions, as they contain only one symbol.

A list $\mathcal{B}$ of all black positions is maintained in increasing order of $i$.

**Generation.** In this stage, all substrings of $w$ with a probability of at least $k$ are generated. For every position $i \in 1 \ldots n$, a list of substrings starting from $i$ is maintained; each substring is associated with two probability values—a value $\pi'$, which is the actual probability of the particular substring, and a temporary value $\pi''$, which is computed by considering all grey positions as white (thus ignoring the probabilities of the gray symbols). The idea at this stage is for every position $i$ to extend the list of substrings generated in the previous position by adding a single symbol at their end every time a white or grey position is met and by creating new substrings every time a black position is met. The generation of new substrings stops whenever a black position is met and the probability $\pi''$ is less than $k$. Notice that the actual substrings may be shorter, as $\pi'$ may have reached the $k$ limit earlier.

**Construction.** All substrings generated in the previous stage now are used to build a generalized suffix tree. As explained earlier, some substrings generated may be longer than the actual substrings to be inserted in the WST because grey positions were considered as white. To correct this, the substrings that are actually longer are pruned to their correct size in the suffix tree.

The weighted suffix tree has various applications, among which are pattern matching, finding repetitions and covers, computing the longest common substring, and extracting motifs, as will be shown in the following sections.

### 8.3.2 Property Suffix Tree

Amir *et al.* [3,4] recently devised the *property suffix tree* (PST), an indexing data structure for property matching on solid strings and showed that it can be built on weighted sequences, too after some preprocessing.

A *property* of a string $x = x[1] \ldots x[n]$ is a set of intervals

$$\mathcal{P} = \{(s_h, f_h) | s_h, f_h \in 1..n \text{ and } s_h \leq f_h\}$$

In the pattern matching problem, a property can be used to limit the matches of the pattern, only to those occurring within the intervals of the given property. Formally, given a text $x = x[1] \ldots x[n]$ and a pattern $y = y[1] \ldots y[m]$, $y$ matches $x[i..j]$ with property $\mathcal{P}$ if $y$ matches $x[i..j]$ and $(s_h, f_h) \in \mathcal{P}$ exists such that $i \geq s_h$ and $j \leq f_h$.

Before describing the PST, a few definitions [4] are in place. For a position $i$ of a string $x = x[1] \ldots x[n]$, let end $(i)$ denote the largest $f_h$ such that $(s_h, f_h) \in \mathcal{P}$ and $i \in s_h \ldots f_h$; end$(i) = NIL$ if no such $f_h$ exists. Consider the suffix tree of $x$, ST$(x)$, and for a node $u$ of ST$(x)$, let $S_u^{\mathcal{P}}$ denote the maximal set of locations $\{i_1, \ldots, i_\ell\} \subseteq \{1, \ldots, n\}$ such that, for every $i_j \in S_u^{\mathcal{P}}$, the following two conditions hold: (i) the leaf of ST$(x)$ that corresponds to the suffix $i$ is a descendant of $u$, and (ii) if end$(i_j) \neq NIL$, then end$(i_j) - i_j > |$label$(u)|$, where label$(u)$ denotes the label on the path from the root to $u$. That is, for every node $u$, a list is maintained of all suffixes prefixed by label$(u)$ that occur within an interval large enough to fit $u$. If $v$ is the parent node of $u$ in ST$(x)$, then it can be established easily that $S_u^{\mathcal{P}} \subseteq S_v^{\mathcal{P}}$. If leaf$(i)$ denotes the leaf node of ST$(x)$ that corresponds to $x[i..n]$, then the path from the root to leaf$(i)$ can be split into two subpaths: (i) the path consisting of all nodes $u$ for which $i \in S_u^{\mathcal{P}}$, and (ii) the path of all nodes $u$ for which $i \notin S_u^{\mathcal{P}}$. Finally, if $v$ is the deepest node in the first path, then let loc$(i)$ either denote the node $v$, when end$(i) - i = |$label$(v)| - 1$ or end$(i) = NIL$, or otherwise loc$(i)$ denotes the edge that connects the two paths.

The PST is then the suffix tree of $x$ modified so that every suffix $x[i \ldots n]$ is moved up to loc$(i)$. Initially, a normal suffix tree of $x$ is constructed [32,43,45]. Then, for every suffix $x[i \ldots n]$, loc$(i)$ is identified, and for every node $v$ (for every edge $e$), a list suf$(v)$ (suf$(e)$) is maintained of all positions $i$ with loc$(i) = v$ (loc$(i) = e$). Then ST$(x)$ is scanned and each node $u$ is marked if either suf$(u)$ is not empty or $u$ is connected to an edge $e$ with suf$(e)$ nonempty, or $u$ is an ancestor of a marked node. All nodes that are not marked at the end of this process are deleted from the tree, and nonbranching paths in the remaining tree are compressed. This process can be performed in $O(n \log |\Sigma| + n \log \log n)$ time.

The PST can be applied to weighted sequences as follows. The weighted sequence $w = w[1] \ldots w[n]$ is preprocessed in a similar manner as in the construction of the WST (Section 8.3.1); positions are marked as solid (white), leading (grey), and branching (black). Then, leading positions are considered as solid, and maximal (in length) factors with probability greater than or equal to $k$ are identified. The number of such factors in $w$, as well as their total length is proved again in [4] to be linear. Finally, all these (extended) maximal factors are concatenated into one solid string, and the probability of each is maintained separately. Then a PST can be built on this solid string with property $\mathcal{P}$ defined on intervals $(s_h, f_h)$ such that the probability of the factor starting at position $s_h$ and ending at $f_h$ is at least $k$, and this factor is maximal (extending to the left or right would yield probability less than $k$). Because the total length of the extended factors is linear, the PST of a weighted sequence can be built in $O(n \log |\Sigma| + n \log \log n)$ time.

It should be noted that, in the following sections, whenever we demonstrate how the WST can be used to solve a problem in weighted sequences, the PST can be applied equally well.

## 8.4 PATTERN MATCHING

The *pattern matching* problem, in its simplest form, can be defined as the problem of locating all occurrences of a given string, the pattern, within a usually longer string, the text. Numerous variants of this problem exist, such as allowing gaps between the symbols, allowing do-not-care symbols in either the text or the pattern (or both), and so on. For solid strings, there are literally hundreds of algorithms for all types of pattern matching (see, for instance, [15,21,41] for an overview).

The problem of exact pattern matching on weighted sequences is defined formally as follows: given a string $y = y[1] \ldots y[m]$ (the pattern), a weighted sequence $w = w[1] \ldots w[n]$ (the text), and a constant $k \in 0..1$, compute the set, $\mathcal{I}$, of positions in $w$ where $y$ occurs with a probability of at least $k$

$$\mathcal{I} = \{i \in 1 \cdots n | \; \Pi_i(y) \geq k\}$$

### 8.4.1 Pattern Matching Using the Weighted Suffix Tree

Similar to the traditional suffix trees, the weighted suffix tree can be used to perform pattern matching on weighted sequences. The algorithm is simple; first the WST is built from the weighted sequence $w$ for the given threshold probability k. Then, starting from the root of the WST, the pattern $y$ is spelt out following the corresponding path on the tree. If a leaf is reached before the pattern has been spelt out completely, then there is no occurrence of $y$ in $w$ with probability at least $k$. On the other hand, if a node $v$ is reached, then there is an occurrence of $y$ in all positions $i$ that are stored in the leaves in the subtree rooted at $v$ (or $v$ itself if $v$ is a leaf). The running time of this algorithm is $O(n)$ for preprocessing the weighted sequence and is $O(m + |\mathcal{I}|)$ for locating occurrences of the pattern, where $|\mathcal{I}|$ denotes the number of occurrences of $y$ in $w$ with probability at least $k$.

Clearly, this algorithm is beneficial when the same weighted sequence is searched repeatedly for different patterns. However, it must be noted that the time to construct and the size of the WST is heavily based on the choice of the threshold probability $k$; it is crucial, therefore, that $k$ is a small constant. Otherwise, the time and space cost is prohibitive. Moreover, if the value of $k$ varies across different queries, then a new WST must be built for each such $k$, making this algorithm impractical for these cases.

### 8.4.2 Pattern Matching Using Match Counts

In this section, we present an alternative algorithm [11] for the pattern matching problem, which although slower than the previous one, has the advantage of not depending on the value of $k$. The algorithm operates in two stages; it first locates all positions of $w$ that match $y$, regardless of their probability; then, the probabilities of the matches are computed, and positions $i$ with $\Pi_i(y) < k$ are discarded.

At the heart of the algorithm lies the *match count* problem [21]. For a weighted sequence $w = w[1] \dots w[n]$ and a string $y = y[1] \dots y[m]$, the match count problem can be defined as follows: compute vector $M(w, y)$, such that $M(w, y)[i]$, for $i \in 1 \dots n$, is the number of symbols of $y$ that match with symbols in $w$, when $y[1]$ is aligned with $w[i]$. Clearly, the positions, $i$, for which $M(w, y)[i] = |y|$ are precisely the positions where $y$ matches $w$.

The problem can be tackled by considering $|\Sigma|$ subproblems, one for each symbol $a \in \Sigma$. Let $M_a(w, y)$ be a vector such that $M_a(w, y)[i]$ denotes the number of $a$s of $w$ that match with $a$s of $y$, when $y[1]$ is aligned with $w[i]$, for $i \in 1 \dots n$. It is straightforward to see that

$$M(w, y) = \sum_{\forall a \in \Sigma} M_a(w, y)$$

To compute each $M_a(w, y)$, two bit vectors, $\bar{y}_a$ and $\bar{w}_a$, are constructed in which the $i$-th position is 1 if $a$ occurs in $y[i]$ and $w[i]$, respectively, and 0 otherwise. Then, the number of matches for the symbol $a$, when $y[1]$ is aligned with position $w[i]$ can be computed as follows:

$$M_a(w, y)[i] = \sum_{j=1}^{m} \bar{y}_a[j] \times \bar{w}_a[i + j - 1]$$

where $\bar{w}_a[i + j]$ is considered to be zero for all $i + j > n$. But this sum of products is nothing more than the vector correlation of $\bar{y}_a$ and $\bar{w}_a$ and thus can be computed in $O(n \log m)$ time using the fast fourier transform (FFT) [13].

At the second stage, the probabilities of the occurrences are computed, using an algorithm similar to that of stage one and the observation that

$$\Pi_i(y) = \prod_{j=1}^{m} \pi_{i+j-1}(y[j]) \geq k \text{ if and only if } \sum_{j=1}^{m} \log(\pi_{i+j-1}(y[j])) \geq \log k$$

Specifically, we construct new vectors $\hat{y}_a$ and $\hat{w}_a$ such that

$$\hat{y}_a[j] = \begin{cases} 1 & \text{if } y[j] = a \\ 0 & \text{otherwise} \end{cases} \qquad \hat{w}_a[i] = \log(\pi_i(a))$$

for all $j \in 1 \ldots m$ and $i \in 1 \ldots n$. Then, similarly to stage one, the sums of the probabilities of the symbols of $y$ for each possible alignment of $y$ with $w$ can be computed by performing $|\Sigma|$ FFTs as follows:

$$\hat{M}(w, y) = \sum_{\forall a \in \Sigma} \hat{M}_a(w, y) \quad \text{where} \quad \hat{M}_a(w, y)[i] = \sum_{j=1}^{m} \hat{y}_a[j] \times \hat{w}_a[i + j - 1]$$

The vectors $M(w, y)$ and $\hat{M}(w, y)$ can be used in conjunction to identify all occurrences of $y$ in $w$ with a probability greater than or equal to $k$. The running time of this algorithm is $O(|\Sigma| n \log m)$ because at each stage, $|\Sigma|$ vector correlations are performed with the use of fast fourier transform. If the number of occurrences identified in stage one is less than $(n \log m)/m$, then stage two can be skipped, and the probabilities can be computed straightforwardly in $O(n \log m)$ time (thus avoiding $|\Sigma|$ FFTs, which are costly operations).

### 8.4.3 Pattern Matching with Gaps

This section is concerned with the problem of pattern matching with gaps—allowing gaps between occurrences of successive symbols of the pattern $y$ in the weighted sequence $w$. Formally, given a string $y = y[1] \ldots y[m]$, a weighted sequence $w = w[1] \ldots w[n]$, a constant $k \in 0 \ldots 1$, and a constant $\alpha$, the $\alpha$-bounded pattern matching with gaps problem is to find all positions $i_1 \in 1 \ldots n$ such that positions $i_2, i_3, \ldots, i_m$ exist for which

- $\pi_{i_j}(y[j]) > 0$, for all $j \in 1 \ldots m$
- $\prod_{j=1}^{m} \pi_{i_j}(y[j]) \geq k$
- $i_j - i_{j-1} - 1 \leq \alpha$, for all $j \in 2 \ldots m$

For example, let $y = \texttt{TGA}$,

$$w = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0.30_{\texttt{A}} & 0.25_{\texttt{A}} & 0.00_{\texttt{A}} & 0.40_{\texttt{A}} & 0.80_{\texttt{A}} & 0.00_{\texttt{A}} \\ 0.00_{\texttt{C}} & 0.25_{\texttt{C}} & 1.00_{\texttt{C}} & 0.20_{\texttt{C}} & 0.05_{\texttt{C}} & 0.50_{\texttt{C}} \\ 0.20_{\texttt{G}} & 0.50_{\texttt{G}} & 0.00_{\texttt{G}} & 0.20_{\texttt{G}} & 0.10_{\texttt{G}} & 0.00_{\texttt{G}} \\ 0.50_{\texttt{T}} & 0.00_{\texttt{T}} & 0.00_{\texttt{T}} & 0.20_{\texttt{T}} & 0.05_{\texttt{T}} & 0.50_{\texttt{T}} \end{pmatrix} \end{array}$$

$k = 0.05$, and $\alpha = 1$. Then $y$ occurs in $w$ at position 1, with $\alpha$-bounded gaps, because

$$\Pi_1(y) = \pi_1(\texttt{T}) \times \pi_2(\texttt{G}) \times \pi_4(\texttt{A}) = 0.5 \times 0.5 \times 0.4 = 0.1 \geq k$$

Notice the gap (of size 1) between the occurrences of $y[2]$ and $y[3]$.

The pattern matching with gaps problem is tackled in [11]. The idea behind this algorithm is to use dynamic programming to locate continuously increasing prefixes of $y$ in $w$, allowing gaps between the occurrences of the symbols of $y$ in $w$. Two dynamic programming tables are used, one for finding the occurrences and one for computing the probabilities of these occurrences. As it is more natural for this algorithm, it is reporting the *ending* positions of occurrences rather than the starting positions. Moreover, notice that because of the gaps, it is possible that two or more occurrences of $y$ end at the same position $i$ in $w$. The algorithm is only considering for each position $i$ in $w$, at most one occurrence of $y$ ending at $i$; if two or more such occurrences exist, then only the one with the maximum probability will be considered, provided that its probability is larger than or equal to $k$.

Let $D$ be an $(n + 1) \times (m + 1)$ matrix. $D[i, j]$, for $i \in 1 \ldots n$ and $j \in 1 \ldots m$, indicates whether there is an occurrence of $y[1 \ldots j]$ ending at position $i$ of $w$, allowing gaps of size at most $\alpha$ and with probability larger than or equal to $k$. In particular, $D[i, j]$ will contain $\ell$, if $\ell$ is the last (rightmost) position of $w$ where the $j$-th prefix of $y$ ends, with adequate probability and without violating the gap invariant; $D[i, j] = -1$ otherwise. The base conditions for $D$ are $D[i, 0] = i$, for all $i \in 0 \ldots n$, and $D[0, j] = -1$ for all $j \in 1 \ldots m$.

The value of $D[i, j]$, for $i \in 1 \ldots n$ and $j \in 1 \ldots m$, is determined by the following three cases:

1. $y[j]$ matches $w[i]$ and the previous prefix $y[1 \ldots j - 1]$ ends at a position $i' < i$ of $w$ such that $i - i' - 1 \leq \alpha$. In this case, the gap between $y[j - 1]$ and $y[j]$ (if there was such a gap) is closed. Note that, for this case to be true, one more condition has to be satisfied: the probability of $y[1 \ldots j]$ must be at least $k$. We defer the discussion of the probabilities until later in this section.
2. There is no occurrence of $y[j]$ in $w[i]$, but the same prefix $y[1 \ldots j]$ ends at some position $i' < i$ such that $i - i' \leq \alpha$, and thus, the gap between $y[j]$ and $y[j + 1]$ can be extended (or opened, if it did not exist already).
3. None of the aforementioned conditions is true, and therefore, there cannot be a match of $y[1 \ldots j]$ ending at position $i$.

These three cases are summarized in the following recurrence relation:

$$D[i, j] = \begin{cases} j, & \begin{array}{lll} \text{if} & y[j] \text{ occurs in } w[i] & \text{and} \\ & D[i-1, j-1] \geq 0 & \text{and} \\ & i - D[i-1, j-1] - 1 \leq \alpha & \text{and} \\ & \max(L[i-1, j-1]) \cdot \pi_i(y[j]) \geq k & \end{array} \\ D[i-1, j], & \begin{array}{ll} \text{if} & \text{previous case does not hold and} \\ & i - D[i-1, j] - 1 < \alpha \end{array} \\ -1, & \text{otherwise} \end{cases} \tag{8.2}$$

where $L[i, j]$ denotes the (maximum) probability of $y[1 \ldots j]$ ending at position $i$ of $w$.

Recall that more than one occurrence of $y[1..j]$ might exist ending at (or before) position $i$ of $w$. Consequently, each $L[i, j]$ should be a data structure able to store at most $\alpha$ probabilities, $\Pi_{i-\alpha+1}, \Pi_{i-\alpha+2}, \ldots, \Pi_i$, where $\Pi_h$, $h \in i - \alpha + 1 \ldots i$, denotes the maximum probability among the probabilities of the occurrences of $y[1 \ldots j]$ that end precisely at position $h$ (and not before), provided that this probability is greater than or equal to the cut-off probability $k$. Moreover, two adjacent cells $L[i-1, j]$ and $L[i, j]$ should have the same values with the following exceptions:

- The first element of $L[i-1, j]$, the maximum probability of the occurrences ending at position $i - \alpha$ (if any such occurrence exists), should not appear in $L[i, j]$ because this element is no longer satisfying the gap invariant.
- A new element, the maximum probability of the occurrences ending at position $i$ (if any such occurrence exists), must be inserted into $L[i, j]$. This can be computed by multiplying $\pi_i(y[j])$ with the maximum probability of the occurrences of $y[1 \ldots j-1]$ ending at (or before) position $i-1$.

Following these observations on the properties of $L$, it was established in [11] that if no trace-back of the dynamic programming is required, then a column-wise computation of $L$ (and $D$) would allow a single data structure, $L$, to be used per column, and so, at iteration $i$ of the algorithm, $L$ is simply *updated* from that of iteration $i-1$. In this case, a heap-ordered queue [18] can be used for $L$, which supports the update of $L$ from one iteration to the next in constant time.

If, on the other hand, trace-back is required, to identify the actual occurrences of $y$ in $w$, then at iteration $i$ (for a given $j$) the whole list $L[i-1, j]$ must be *copied* into $L[i, j]$, and possibly, its head and tail must be updated as explained. In this case, a simplified version of a persistent list [26] can be used, which supports amortized constant time of the computation of $L[i, j]$ from $L[i-1, j]$.

The running time of the algorithm is $O(mn)$, assuming that no trace-back is required, because there are $mn$ iterations, and at each iteration $(i, j)$, the computation of $D[i, j]$ and the update of $L[i, j]$ (from $L[i-1, j]$) can be performed in constant time.

### 8.4.4 Pattern Matching with Swaps

A *swap* in a string is the interchange of two symbols appearing in consecutive positions. A string $y'$ is a swapped version of a string $y = y[1] \ldots y[m]$ if and only if $y'$ is derived from $y$ by interchanging some symbols appearing in consecutive positions in $y$ ($y'[j] = y[j+1]$ and $y'[j+1] = y[j]$), where each position $j$ of $y$ can participate in at most one swap operation. The pattern matching with swaps problem is that of locating all positions $i$ in a weighted sequence $w = w[1] \ldots w[n]$ in which any swapped version of the pattern $y$ occurs with probability $k$ or more.

This problem has been addressed in [47] by combining the techniques presented in [22, 23, 29] for building the weighted suffix tree with the algorithms presented in [5] for solving the swap-matching problem in solid strings. In particular, first all (solid) factors of $w$ with a probability of at least $k$ are identified, using the steps described in Section 8.3.1. The number of such factors is proven to be $O(n)$ for a fixed value of $k$. Next these factors are concatenated to a long (solid) string $x$, and the starting positions (within this new string) of the factors are stored in a separate array. In this way, both the pattern, $y$, and the new text, $x$, are solid strings, and therefore, the overlap matching [5] technique can be applied to find all matches with swaps.

The preprocessing step of extracting the maximal factors takes $O(n)$ time. Therefore, the running time of this algorithm is dominated by the overlap match algorithm, which takes $O(n \log m \log \sigma)$, where $\sigma = \min(m, |\Sigma|)$.

## 8.5 APPROXIMATE PATTERN MATCHING

*Approximate pattern matching* refers to locating factors of a weighted sequence $w$ that are *similar*, rather than identical, to a given pattern $y$. The notion of similarity can be defined via a *distance function*, which computes how dissimilar two strings are; the larger the distance, the smaller the similarity. The distance between two strings can be viewed as the number of *errors* that supposedly transformed one string to the other. In many cases, a maximum distance $d$ is provided as input to the problem so that only approximate occurrences with distance at most $d$ are located.

Amir *et al.* [6] devised algorithms for the approximate pattern matching problem on weighted sequences under the *hamming distance* [46] and briefly addressed the problem under the *edit distance* [28]. Because of space limitations, we only cover the former here.

### 8.5.1 Hamming Distance

The *Hamming distance* between two equal-length[6] strings is defined as the minimum number of *substitutions* that are necessary to transform one string into the other. In solid strings, the Hamming distance is symmetric in the sense that it does not make a difference whether the error is assumed to occur in the text or in the pattern. However, when comparing a solid string with a weighted sequence this is not the case; depending on whether the errors are assumed to occur in the weighted sequence or in the string, the definition of the distance function differs, with consequences for the approximate pattern matching algorithms. Here, both cases are examined.

***8.5.1.1 Hamming Distance when Errors Occur in the Text.*** Consider a string $y = y[1] \dots y[m]$ and an equal-length weighted sequence $w = w[1] \dots w[m]$. The Hamming distance, under the assumption that the string is error-free, and thus, all errors occur in the weighted sequence, can be defined as the minimum number of positions in $w$ that must be substituted by a single symbol (*i.e.*, with probability 1)

[6] The Hamming distance is defined only on strings of equal length.

so that that $y$ matches $w$ with a probability of at least $k$. From this definition, it is clear that if one substitutes at most $m$ positions in $w$ with the corresponding symbols from $y$, then $y$ will match $w$ regardless of how close to 1 $k$ is.

The approximate pattern matching problem when using the Hamming distance with errors in the weighted sequence can be defined as follows: given a weighted sequence $w = w[1] \dots w[n]$, a string $y = y[1] \dots y[m]$ and a constant $k \in 0 \dots 1$, identify for each position $i \in 1..n - m + 1$ of $w$ the minimum number of errors, $d_i$, that must be substituted in $w$ for $y$ to match with the substring $w[i \dots i + m - 1]$ of $w$ with a probability greater than or equal to $k$.

The first step to solve this problem is to transform the input as follows:

- A string $w' = w'[1] \dots w'[|\Sigma|n]$ of nonpositive integers is constructed from $w$, where $|\Sigma|$ consecutive positions of $x$ correspond to one position of $w$ as follows:

$$w'[(i - 1)|\Sigma| + h] = \log \pi_i(a_h), \text{ for all } h \in 1..\Sigma$$

  In simple words, for every position $i$ of $w$, the logarithms of the probabilities of all symbols are written one after the other
- A string $y' = y'[1] \dots y'[|\Sigma|m]$ of bits is constructed from $y$ as follows:

$$y'[(i - 1)|\Sigma| + h] = \begin{cases} 1, & \text{if } y[i] = a_h \\ 0, & \text{otherwise} \end{cases}$$

- A new threshold probability is defined $k' = \log k$

With this transformation in place, the approximate pattern matching problem can be reduced to the *ignored mask bits* problem, which is defined as follows: given a string $w' = w'[1] \dots w'[|\Sigma|n]$ of nonpositive integers, a string $y' = y'[1] \dots y'[|\Sigma|m]$ of bits and a nonpositive constant $k'$, find for every location $i' \in 1..|\Sigma|n$ the minimum number, $d_{i'}$, of bits in $y'$ that must change from 1 to 0 so that $\sum_{j=1}^{m} w'[i' + j]y'[j] \geq k'$. Then the minimum distance, $d_i$, for the $i$th position of the original sequence $w$ is simply $d_i = \lceil d_{i'}/|\Sigma| \rceil$.

The algorithm that solves the ignored mask bits problem [6] uses divide-and-conquer to split the input into subproblems that can be tackled by two special cases (i) the *bounded alphabet* case in which the size of the alphabet of $w'$ is bounded, and (ii) the *bounded relevant numbers* case in which the number of elements in $w'$ that are greater than $k'$ is bounded. The former problem can be solved in $O(rn \log m)$ time, where $r$ is the upper bound on the size of the alphabet, and the latter in $O(ns)$ time, where $s$ is the maximum number of elements larger than $k'$ in any substring of $w'$ of length $m$. The divide-and-conquer algorithm sorts the text elements and then splits them into $r$ blocks of size at most $2|\Sigma|\frac{m}{r}$ each. In this way, the bounded alphabet algorithm and the bounded relevant numbers algorithm are applied on each block. To achieve optimal time, the number of blocks, $r$, is proven [6] to be

$$r = \sqrt{\frac{m}{\log m}} = \frac{m}{\sqrt{m \log m}}$$

which yields overall running time

$$O(rn \log m + n\frac{m}{r}) = O(n\sqrt{m \log m})$$

### *8.5.1.2 Hamming Distance when Errors Occur in the Pattern.*

The Hamming distance between $y$ and $w$, when errors are assumed to occur in $y$, can be defined as the minimum number of symbols in $y$ that must be substituted by other symbols for $y$ to match $w$ with probability at least $k$. In contrast to the previous definition of the Hamming distance, here it might be the case that $y$ cannot match $w$ even if all $m$ symbols of $y$ are substituted.

Let $y$ and $w$ be of length $m$ and consider the probability of $y$ occurring at $w[1]$ as less than $k$

$$\Pi_1(y) = \prod_{j=1}^{m} \pi_j(y[j]) < k$$

The aim is to substitute as few symbols from $y$ as possible, until $\Pi_1(y) \geq k$. Clearly, if a symbol $y[j]$ is to be substituted, then it will be replaced by the symbol with the maximum probability at position $j$ of $w$ (*i.e.*, $\max_{h=1}^{|\Sigma|}(\pi_j(a_h))$). To find the *minimum* number of such positions $j$ to be substituted, one has to order the positions according to the *gain* each of them offers to the overall probability $\Pi_1(y)$. The gain for substituting $y[j]$ is the ratio of the maximum probability at position $j$ of $w$ divided by the probability of the symbol being replaced

$$\frac{\max_{h=1}^{|\Sigma|}(\pi_j(a_h))}{\pi_j(y[j])} \tag{8.3}$$

Consequently, it suffices to sort the input sequences in decreasing order of their ratio (8.3) and keep substituting symbols from $y$ in this order until $\Pi_1(y) \geq k$. As highlighted in [6], substituting a symbol $y[j]$ with a new symbol $a$ is equivalent to substituting, at position $j$, *in the weighted sequence* the probability $\pi_j(y[j])$ with $\pi_j(a)$.

The algorithm for solving the approximate pattern matching problem, under the Hamming distance function with errors in the pattern, sorts the weighted sequence according to the ratio (8.3) and then splits it into $O(\frac{m}{\sqrt{m \log m}})$ blocks of size $O(\sqrt{m \log m})$, similar to the algorithm that considers errors in the text. Then, for each location $i$, the probabilities are computed $O(\frac{m}{\sqrt{m \log m}})$ times as follows: In the first iteration, the probabilities are computed without any substitutions in the text; in the second iteration, the probabilities are computed having substituted the element with the highest gain ratio in each group; $\ldots$; in the $j$-th iteration, the probabilities are computed having substituted the $j-1$ elements with the highest gain ratio. Each calculation can be done in $O(n \log m)$ time using FFT; thus, the overall running time is $O(n\sqrt{m}\log m)$.

## 8.6 REPETITIONS, COVERS, AND TANDEM REPEATS

In this section, we examine algorithms for locating repetitive elements in weighted sequences—that is, factors that occur two or more times within a weighted sequence. Again, various definitions are adopted according to the underlying assumptions on what constitutes a repetition.

Consider two occurrences of a factor $y = y[1]\ldots y[m]$ in a weighted sequence $w = w[1]\ldots w[n]$, such that the distance between the two occurrences is less than $m$; in contrast to solid strings, here the two occurrences do *not* necessarily overlap with each other. The following example illustrates this:

$$w = \begin{array}{cccccc} \scriptstyle 1 & \scriptstyle 2 & \scriptstyle 3 & \scriptstyle 4 & \scriptstyle 5 & \scriptstyle 6 \\ \end{array} \begin{pmatrix} 0.30_{\mathtt{A}} & 0.25_{\mathtt{A}} & 0.20_{\mathtt{A}} & 0.40_{\mathtt{A}} & 0.80_{\mathtt{A}} & 0.00_{\mathtt{A}} \\ 0.00_{\mathtt{C}} & 0.25_{\mathtt{C}} & 0.80_{\mathtt{C}} & 0.20_{\mathtt{C}} & 0.20_{\mathtt{C}} & 0.50_{\mathtt{C}} \\ 0.20_{\mathtt{G}} & 0.50_{\mathtt{G}} & 0.00_{\mathtt{G}} & 0.20_{\mathtt{G}} & 0.00_{\mathtt{G}} & 0.00_{\mathtt{G}} \\ 0.50_{\mathtt{T}} & 0.00_{\mathtt{T}} & 0.00_{\mathtt{T}} & 0.20_{\mathtt{T}} & 0.00_{\mathtt{T}} & 0.50_{\mathtt{T}} \end{pmatrix} \tag{8.4}$$

$y = \mathtt{ACC}$ occurs at positions 1 and 2; however, the two occurrences do not overlap with each other: Rather, they are making use of different symbols for some of the common positions; specifically at position 2, the first occurrence is using C, whereas the second uses A.

Depending on what the weighted sequence, or the repetitions themselves, represent, this “anomaly” might or might not be acceptable. For this reason, the following types of repetitions were defined in [9]:

- *Simple repetitions* are repetitions in which “borderless” overlaps are allowed.
- *Strict repetitions* introduce the extra restriction that distinct occurrences should be using the same symbol for every common position in $w$.

The same distinction applies to *covers*. Recall that a cover of length $\ell$ is a repetition whose consecutive occurrences are no more that $\ell$ distance from each other, and the first (last) occurrence starts (ends) at the first (last) position of $w$ (thus “covering” $w$). Therefore, simple and strict covers can be defined similarly. For instance, in the previous weighted sequence, the string ACC is a simple cover (occurs at positions 1, 2, and 4) whereas ACAC is a strict cover (occurs at positions 1 and 3) and the two occurrences do overlap with each other.

In this section, we present the algorithms for locating repetitions in a weighted sequence. Currently, the only linear time algorithm is based on the use of the weighted suffix tree (see Section 8.6.1). However, because of the inherent drawbacks associated with the WST, such as the large size and the dependence on $k$, alternative algorithms have been discovered, which, although they do not achieve the optimal asymptotic running time, in practice they likely to perform better; these are presented in Sections 8.6.2–8.6.5. These algorithms are not perfect either, their main drawback being that they compute repetitions of prespecified length only.

### 8.6.1 Finding Simple Repetitions with the Weighted Suffix Tree

The asymptotically optimal running time for this problem was achieved by Iliopoulos *et al.* [22], in which the authors used the weighted suffix tree [29] to find the repetitions. The algorithm works as follows: once the WST of weighted sequence $w = w[1] \ldots w[n]$ is built (see Section 8.3.1), it is traversed bottom-up, and at each internal node $v$, the factors that are stored in the leaves under $v$ are reported. The number of factors in the WST is linear, and thus, the running time is also linear.

### 8.6.2 Fixed-Length Simple Repetitions

The problem of finding all simple repetitions of a weighted sequence is defined as follows: given a weighted sequence $w = w[1] \ldots w[n]$, a constant $k \in 0 \ldots 1$, and an integer $m \in 1 \ldots n - 1$ identify, for every length-$m$ factor $y$ that occurs at least twice in $w$, the set $\mathcal{I}_y$ of positions where $y$ occurs in $w$ with a probability greater than or equal to $k$.

Christodoulakis *et al.* [10, 12] devised an algorithm for this problem based on ideas first presented in [27] for finding repetitions in solid strings. The idea is to split the positions of the input sequence into *equivalence classes*; two positions that belong to the same equivalence class contain the same factor of length $m$ in which $m$ is a prespecified constant. Then, equivalence classes of factors of increasing length are computed by combining equivalence classes for smaller factors that already have been computed.

Formally, equivalence classes are defined as follows: Two positions $i$ and $j$ of a weighted sequence $w = w[1] \ldots w[n]$ are $m$ equivalent ($m \in 1 \ldots n$ and $i, j \in 1 \ldots n - m + 1$) (written $i E_m j$) if and only if at least one solid string $y$ exists of length $m$ that occurs at both positions $i$ and $j$ with a probability greater than or equal to $k$.

The relation $E_m$, is called an *equivalence relation* and is represented as a vector, $E_m[1 \ldots n - m + 1]$, of sets of integers; the set $E_m[i]$ consists of the integers that represent the equivalence classes of the factors of length $m$ that start at position $i$ and have a probability of at least $k$.

When dealing with solid strings, every position $i \in 1 \ldots n - m + 1$ contains precisely one factor of length $m$. For weighted sequences, on the other hand, any position may contain more than one symbol; thus, more than one factor of length $m$ can occur at the same position with probability greater than or equal to $k$. Consequently, a single position $i$ of a weighted sequence may belong to more than one equivalence class.

To solve the problem of locating repeated substrings of length $m$ over a weighted sequence $w$, the algorithm proceeds as follows; it first computes the equivalence relation $E_d$:

1. Scan $w$ to construct relation $E_1$.
2. Construct $E_m$ from smaller equivalence relations that already have been computed, using the principles of binary multiplication (binary decomposition of $d$) (*e.g.*, combine $E_1$ with itself to get $E_2$, $E_2$ with itself to get $E_4$, $E_4$ with $E_1$ to get $E_5$, and so on).

Then, by a single scan through $E_m$, one can identify all equivalent positions and thus all repetitions of length $m$.

The first step, constructing $E_1$, is straightforward; scan $w$, and for each symbol at position $i \in 1 \ldots n$, insert into $E_1[i]$ the integer (equivalence class) that represents this symbol. The second step, which is performed repeatedly and is used to construct new equivalence relations from existing ones, is explained next.

Given two equivalence relations, $E_{m_1}$ and $E_{m_2}$, $m_2 \leq m_1$, the relation $E_m$, where $m = m_1 + m_2$, can be constructed using the following observation:

$$i E_m j \quad \text{if and only if} \quad i E_{m_1} j \text{ and } i + m_1 E_{m_2} j + m_1 \tag{8.5}$$

To implement efficiently the construction of $E_m$, two sets of stacks are used: $P(1), \ldots, P(e_{m_1})$ and $Q(1), \ldots, Q(e_{m_2})$, where $e_{m_1}$ and $e_{m_2}$ represent the number of distinct equivalence classes in $E_{m_1}$ and $E_{m_2}$, respectively.

1. Sort vector $E_{m_1}$ using the $P$-stacks: run through $E_{m_1}$, and for each equivalence class $y$ at each position $i$, push $(i, \pi_i(y))$ into $P(y)$. This step sorts the positions according to the equivalence classes of length $m_1$.
2. Resort using the $Q$-stacks: pop each $P(y)$ until it is empty. As position $(i, \pi_i(y))$ is popped from $P(y)$, push $(i, \pi_i(y)\pi_{i+m_1}(x))$ onto $Q(x)$, $x \in E_{m_2}[i + m_1]$, provided that $i + m_1 \leq n - (m_1 + m_2 - 1)$ and $\pi_i(y) \times \pi_{i+m_1}(x) \geq k$. That is, resort the positions of $E_{m_1}$ according to the factors of length $m_2$ that occur $m_1$ positions to the right.
3. Construct $E_m$: pop each $Q$-stack until empty. Let $c$ (initially set to 1) denote the current equivalence class. As each $(i, \pi_i(y)\pi_{i+m_1}(x))$ is popped from a given stack $Q(x)$, store $c$ in the set $E_m[i]$. $c$ is incremented every time a new factor $y$ is popped out of $Q(x)$ or if $y$ is the first factor popped out of $Q(x)$.

The process of combining two equivalence relations to build a larger one takes $O(n)$ time, provided that the input weighted sequence is drawn from a bounded-size alphabet. This is a consequence of the fact that the number of factors occurring at any position with probability $k$ or more is constant, as discussed in Section 8.3.1. Because this process is repeated $\log m$ times, the overall running time of the algorithm is $O(n \log m)$.

### 8.6.3 Fixed-Length Strict Repetitions

Strict repetitions are similar to, but slightly more restricted than, their simple counterparts. We now are looking for sets of positions such that no two occurrences of the same factor occur in overlapping positions without the occurrences themselves overlapping. Formally, given a weighted sequence $w = w[1] \ldots w[n]$, a constant $k \in 0 \ldots 1$, and an integer $m \in 1 \ldots n-1$, identify for every length $m$ simple repetition $y$ in $w$, a set $\mathcal{I}'_y \subseteq \mathcal{I}_y$ of positions in which $y$ occurs in $w$ with a probability greater than or equal to $k$ such that, for every pair of positions $i, i' \in \mathcal{I}'_y$, either $|i - i'| \geq m$ or $m - |j - j'|$ is the length of a border of $y$.

To identify which occurrences of a simple repetition form a strict repetition, one must discover whether any two occurrences of a factor use the same symbols at all their overlapping positions (if any). This can be done by simply computing the borders of the factor under question and by checking whether the right-most occurrence begins at a position that corresponds to a (right) border of the factor.

Consequently, the algorithm presented in Section 8.6.2, must be modified so that, together with an equivalence relation $E_m$, a list $F_m$ of all equivalence classes and factors they correspond to are stored. Initially, the factor list $F_1$ of factors of length 1 (single symbols) is built by a simple scan in $w$. Then, two equivalence relations $E_{m_1}$ and $E_{m_2}$ are combined, and their factor lists $F_{m_1}$ and $F_{m_2}$ are concatenated. Finally, once all *simple* repetitions are found, as in the algorithm of Section 8.6.2, their occurrences are tested to see whether they overlap to identify strict repetitions. The running time of the updated algorithm is $O(nm)$, because now the border arrays [1] of $O(n)$ factors of length $m$ must be computed.

### 8.6.4 Fixed-Length Tandem Repeats

The problem of finding tandem repeats in weighted sequences is defined as follows: given a weighted sequence $w = w[1] \ldots w[n]$, a constant $k \in 0 \ldots 1$, and an integer $m \in 1 \ldots n-1$, identify for every length $m$ simple repetition $y$ in $w$, all sets $\mathcal{I}''_y \subseteq \mathcal{I}_y$ of positions in which $y$ occurs in $w$ with probability greater than or equal to $k$ such that, for every pair of positions $i, i' \in \mathcal{I}''_y$ with $i < i'$, $i' = i + m$.

The algorithms for simple repetitions (Section 8.6.2) can be extended easily to locate tandem repeats of length $m$. The time consumed to construct $E_m$ is $O(n \log m)$, whereas scanning $E_m$ to identify tandem repeats takes linear time, because there is a constant number of factors for every position of $w$, and each is only accessed once.

In [24], a different algorithm was presented for the same problem, which was based on Crochemore's algorithm [14] for finding tandem repeats in solid strings. However, this algorithm later was proved [10] to run in $O(n^2)$ time.

### 8.6.5 Identifying Covers

It is straightforward to see that covers can be identified by checking whether the occurrences of each repetition "cover" the whole of $w$. Therefore, to locate all length $m$ covers, it suffices to:

1. Find all length $m$ repetitions, $y$, and the corresponding positions of occurrence, $\mathcal{I}_y$, using one of the algorithms presented in the previous sections
2. Identify which of these factors have an occurrence at position 1 of $w$ and another occurrence at position $n - m + 1$; all other factors can be discarded as they certainly do not "cover" the whole of $w$
3. For each $y$ selected scan $\mathcal{I}_y$ and check whether the distance between consecutive occurrences of $y_h$ is always less than or equal to $d$

Steps 2 and 3 can be applied in $O(n)$ time; thus, the overall running time is dominated by the algorithm used in step 1 to locate the repetitions.

## 8.7 MOTIF DISCOVERY

In biological sequences, *motifs* are repeating factors in a sequence or set of sequences that have some important biological role. Similar to repetitions, which we examined in the previous section, the exact string of the motif is unknown at first; however, other information about the motif is normally available, which helps distinguish motifs from other random repetitions. For instance, the minimum number of occurrences of the motif in a sequence or set sequences might be known in advance, or the structure of the motif might be known. Moreover, often the repeated factors of a motif are not identical, because of mutations or errors from the sequencing equipment.

### 8.7.1 Approximate Motifs in a Single Weighted Sequence

Iliopoulos *et al.* [25] considered the problem of locating approximate motifs in a weighted sequence using the Hamming distance. More specifically, the problem they addressed was to identify the factors (motifs), $y$, of a given weighted sequence $w = w[1] \ldots w[n]$ that have the following properties:

1. $|y| = m \geq 2$
2. The probability of occurrence of each approximate occurrence of $y$ is at least $k \in 0 \ldots 1$
3. The Hamming distance between the identified motif $y$ and its approximate occurrences in $w$ is at most $e$
4. There are at least $q \geq 2$ *approximate* occurrences of $y$ in $w$, and no two occurrences overlap with each other

where $m$, $q$, $e$, and $k$ are prespecified constants.

The algorithm initially constructs the WST of $w$. Then, following a process similar to that of [38], all motifs of length $m$ are identified on the WST, starting from length one and iteratively extending the current model by one symbol either with a match or by considering a mismatch if the limit $e$ already has not been reached. The difference of this algorithm from that of [38] is that overlapping positions have to be filtered out according to the last constraint set earlier.

Let $L = \{v_1, v_2, \ldots, v_{|L|}\}$ denote the set of all internal nodes of the WST with path label of length $m$. For each node $v \in L$, a sorted list, $v^m$ of all leaves under $v$ is maintained, using van Emde Boas trees [44]. Let $L' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_j}\}$, $L' \subseteq L$, be the set of nodes that constitute a candidate motif $y$ (*i.e.*, nodes whose label is within distance $e$ from y). The last property of the motif, to have at least $q$ occurrences, can be verified easily at this stage by simply summing the number of leaves under all nodes in $L'$, $\sum_{h=1}^{j} |v_{i_h}^m|$.

Checking whether the occurrences overlap can be done as follows: scan the leaves of all nodes in $L'$ to identify the one with the smallest position of occurrence, $i$; next, find the first location, $i'$, such that $i' \geq i + |y| + 1$; and so on, until at least $q$ such occurrences are identified.

Once the WST is constructed, building $L$ and the associated lists $v^m$ for all nodes of $L$ can be done in linear time because the lists are disjoint, and they consist of integers in the range $1 \ldots n$. The process of identifying nonoverlapping occurrences can be performed in $O(q|L| \log\log n)$ time because the lists are stored as van Emde Boas trees. Consequently, the total running time of the algorithm is $O(nV^2(e, m)q \log\log n)$, where $V(e, m)$ is the number of motifs of length $m$ with at most $e$ Hamming distance errors (mismatches).

### 8.7.2 Approximate Common Motifs in a Set of Weighted Sequences

Generalizing the approximate motifs problem to more than one weighted sequence yields the following problem: given a set of weighted sequences $W = \{w_1, w_2, \ldots, w_N\}$, identify the motifs, $y$, that have the following properties:

1. $|y| = m \geq 2$
2. The probability of occurrence of each approximate occurrence of $y$ is at least $k \in 0 \ldots 1$
3. The Hamming distance between the identified motif $y$ and its approximate occurrences in weighted sequences of $W$ is at most $e$
4. Approximate occurrences of $y$ occur in at least $q \geq 2$ distinct sequences from $W$

where $m$, $q$, $e$, and $k$ are prespecified constants.

The algorithm for this problem is very similar to that of Section 8.7.1. A generalized weighted suffix tree is constructed from the set of weighted sequences $W$, and occurrences can be checked for whether they overlap in the same manner as in Section 8.7.1. The main difference is the mechanism that is necessary here for checking that the motif occurs in at least $q$ weighted sequences (condition 4). An integer array

$u_v$ of length $q$ is maintained for each node $v$ to store the index $i$, of the weighted sequence, $w_i$, in which an occurrence of the candidate motif has been found. As soon as this vector gets full, then $q$ occurrences have been found in distinct weighted sequences and any other occurrences are ignored from this point on. The process of filling these arrays is described next.

The suffix tree is traversed in a post-order manner. To compute the vector $u_v$ of a node $v$, either of the following suffices (i) find a child, $v'$ of $v$ whose vector $u_{v'}$ is already full, or (ii) merge the arrays of all children of $v$ omitting repetitive entries. Because any node $v$ has at most $|\Sigma|$ children, this process can be accomplished in $O(|\Sigma|q)$ time (or $O(q)$, because $|\Sigma|$ is a constant). Repeating the process for each of the $O(nN)$ nodes in the generalized WST, and for each of the $V(e, m)$ motifs of length $m$, the total running time becomes $O(nNqV(e, m))$.

## 8.8 CONCLUSIONS

Sequence alignment today is one of the major tools for identifying potential structural and/or functional similarities between DNA sequences. With the rapid growth of data in the last 15 years, the importance of effectively storing representations of this data and efficiently extracting information from them has become immense. Weighted sequences were introduced to fulfill this role and are perhaps one of the most advanced (in terms of the amount of information they hold) models for representing whole sets of aligned sequences.

In this chapter, we provided an overview of the most recent advances in the research field of weighted sequences and more specifically in the cases in which the text is weighted and the pattern, or the repetitions and motifs sought for, are solid strings. We covered the problems of indexing, pattern matching (exact and approximate), repetition finding, and motif extraction. These problems have been the subject of research for at least 30 years as far as solid strings are concerned but only recently have received large attention for weighted sequences. Algorithmically, weighted sequences pose a great challenge as the existing, well-studied, algorithms for solid strings rarely can be applied directly.

Despite the growing number of research outcomes on this field, still numerous open problems need to be addressed. In the topic of indexing, for example, a more space-efficient data structure would be beneficial, especially given the already large size of weighted sequences (perhaps, a weighted suffix array to mimic its counterpart in solid strings); of equal importance would be the ability for this data structure to be independent of the threshold probability or at least capable to get updated dynamically every time the threshold changes. Similarly, in the field of pattern matching, there is an endless list of possible definitions of a match, especially if errors are permitted; it would be interesting to elaborate on the application of other distance functions, such as the edit distance, for example, or even to devise new distance functions more appropriate for the weighted nature of the weighted sequences. Regarding repetitions and motif extraction, different applications demand different models, and the ability to incorporate extra information regarding the structure of the repetitive elements helps distinguish significant sequences from random ones.

Although most research on weighted sequences has focused on their biological applications, it should be noted that model weighted sequences can fit easily into other contexts too. Very recently, Makris *et al.* [31] have used weighted suffix trees to model the navigation history of users in a web site, aiming to predict the usage of each web page on the site.

## REFERENCES

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.
2. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A basic local alignment tool. *J Mol Bio*, 215:403–410, 1990.
3. A. Amir, E. Chencinski, C.S. Iliopoulos, T. Kopelowitz, and H. Zhang. Property matching and weighted matching. In Moshe Lewenstein and Gabriel Valiente, editors, *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 4009 of *Lecture Notes in Computer Science*, Springer, Barcelona, Spain, 2006, pp. 188–199.
4. A. Amir, E. Chencinski, C.S. Iliopoulos, T. Kopelowitz, and H. Zhang. Property matching and weighted matching. *Theor Comput Sci*, 395(2–3):298–310, 2008.
5. A. Amir, R. Cole, R. Hariharan, M. Lewenstein, and E. Porat. Overlap matching. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Society for Industrial and Applied Mathematics. Philadelphia, PA, 2001, pp. 279–288.
6. A. Amir, C.S. Iliopoulos, O. Kapah, and E. Porat. Approximate matching in weighted sequences. In Moshe Lewenstein and Gabriel Valiente, editors, *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 4009 of *Lecture Notes in Computer Science*, Springer, Barcelona, Spain, 2006, pp. 365–376.
7. A. Bairoch. PROSITE: a dictionary of sites and patterns in proteins. *Nucleic Acid Res*, 20:2013–2018, 1992.
8. A. Bairoch and P. Bucher. PROSITE: recent developments. *Nucleic Acid Res*, 22:3583–3589, 1992.
9. M. Christodoulakis. *Regularities on Fixed and Weighted Sequences*. PhD dissertation, Department of Computer Science, King's College London, 2005.
10. M. Christodoulakis, C.S. Iliopoulos, L. Mouchard, K. Perdikuri, A. Tsakalidis, and K. Tsichlas. Computation of repetitions and regularities of biologically weighted sequences. *J Computat Bio*, 13(6):1214–1231, 2006.
11. M. Christodoulakis, C.S. Iliopoulos, L. Mouchard, and K. Tsichlas. Pattern matching on weighted sequences. In Katia S. Guimarães and Marie-France Sagot, editors, *Proceedings of the Algorithms and Computational Methods for Biochemical and Evolutionary Networks (CompBioNets)*. King's College London, 2004, pp. 17–30.
12. M. Christodoulakis, C.S. Iliopoulos, K. Tsichlas, and K. Perdikuri. Searching for regularities in weighted sequences. In Theodore Simos and George Maroulis, editors, *Proceedings of the International Conference of Computational Methods in Sciences and Engineering (ICCMSE)*, Lecture Series on Computer and Computational Sciences, 2004, pp. 701–704.

13. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd edition. The MIT Press, Boston, MA, 2001.
14. M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf Process Lett*, 12(5):244–250, 1981.
15. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, Inc., New York, NY, 1994.
16. G.E. Crooks, G. Hon, J.-M. Chandonia, and S.E. Brenner. WebLogo: A sequence logo generator. *Genome Res*, 14:1188–1190, 2004.
17. European Bioinformatics Institute (EMBL-EBI). ClustalW. http:// www.ebi.ac.uk/ clustalw/.
18. H. Gajewska and R.E. Tarjan. Deques with heap order. *Inf Process Lett*, 22(4):197–200, 1986.
19. A.J. Gibbs and G.A. McIntyre. The diagram, a method for comparing sequences. *Eur. J Biochem*, 16(1):1–11, 1970.
20. M. Gribskov, A.D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proc Natl Acad Sci*, 84:4355–4358, 1987.
21. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, MA, 1997.
22. C.S. Iliopoulos, C. Makris, Y. Panagis, K. Perdikuri, E. Theodoridis, and A. Tsakalidis. Computing the repetitions in a weighted sequence using Weighted Suffix Trees. *European Conference on Computational Biology (ECCB), Posters' Track*, 2003, pp. 539–540.
23. C.S. Iliopoulos, C. Makris, Y. Panagis, K. Perdikuri, E. Theodoridis, and A. Tsakalidis. Efficient algorithms for handling molecular weighted sequences. In J.J. Levy, E.W. May, and J.C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics*. Kluwer Academic Publishers, New York, 2004, p. 265.
24. C.S. Iliopoulos, L. Mouchard, K. Perdikuri, and A. Tsakalidis. Computing the repetitions in a weighted sequence. In M. Šimánek, editor, *Proceedings of the 8th Prague Stringology Conference (PSC)*. 2003, pp. 91–98.
25. C.S. Iliopoulos, K. Perdikuri, E. Theodoridis, A. Tsakalidis, and K. Tsichlas. Motif extraction from weighted sequences. *11th International Conference String Processing and Information Retrieval (SPIRE)*, volume 3246, Padova, Italy, 2004, pp. 286–297.
26. H. Kaplan, C. Okasaki, and R.E. Tarjan. Simple confluently persistent catenable lists. *SIAM J Comput*, 30(3):965–977, 2000.
27. R.M. Karp, R.E. Miller, and A.L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing (STOC)*, ACM, New York, NY, 1972, pp. 125–136.
28. V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Sov Phys Dokl*, 10:707–710, 1966.
29. C.S. Liopoulos, C. Makris, Y. Panagis, K. Perdikuri, E. Theodoridis, and A. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundamenta Informaticae*, 71(2,3):259–277, 2006.
30. D.J. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.
31. C. Makris, Y. Panagis, E. Theodoridis, and A.K. Tsakalidis. A web-page usage prediction scheme using weighted suffix trees. In Nivio Ziviani and Ricardo A. Baeza-Yates, editors, *Proceedings of the 14th International Symposium on String Processing and Information*

*Retrieval (SPIRE)*, volume 4726 of *Lecture Notes in Computer Science*, 2007, pp. 242–253.

32. E.M. McCreight. A space-economical suffix tree construction algorithm. *J ACM*, 23(2):262–272, 1976.
33. B. Morgenstern, A. Dress, and T. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc Natl Acad Sci*, 93:12098–12103, 1996.
34. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48:443–453, 1970.
35. W.R. Pearson. Effective protein sequence comparison. *Meth Enzymol*, 266:227–258, 1996.
36. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. In *Proc Natl Acad Sci*, 85:2444–2448, 1988.
37. C. Pizzi and E. Ukkonen. Fast profile matching algorithms – a survey. *Theor Comput Sci*, 395(2–3):137–157, 2008.
38. M.-F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics (LATIN)*, volume 1380 of *Lecture Notes in Computer Science*, Springer, New York, 1998, pp. 374–390.
39. T.D. Schneider and R.M. Stephens. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Res*, 18:6097–6100, 1990.
40. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147:195–197, 1981.
41. W.F. Smyth. *Computing Patterns in Strings*. Addison-Wesley, Reoding, MA, 2003.
42. J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acid Res*, 22:4673–4680, 1994.
43. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
44. P. van Emde Boas. Preserving order in a forest in less than logarithmic time. *Inf Process Lett*, 6(3):80–82, 1977.
45. P. Weiner. Linear pattern matching algorithms. *Proceedings of the 14th Annual Symposium on Switching and Automata Theory*, 1973, pp. 1–11.
46. R.W. Hamming. Error detecting and error correcting codes. *Bell Syst Tech J*, 29(2):147–160, 1950.
47. H. Zhang, Q. Guo, and C.S. Iliopoulos. String matching with swaps in a weighted sequence. *Proceedings of International Symposium on Computational and Information Sciences (CIS)*, Shanghai, China, 2004.

# 9

# DNA COMPUTING FOR SUBGRAPH ISOMORPHISM PROBLEM AND RELATED PROBLEMS

Sun-Yuan Hsieh, Chao-Wen Huang, and Hsin-Hung Chou

## 9.1 INTRODUCTION

A DNA is a polymer made up of a sequence of subunits known as *nucleotides*. Distinct nucleotides are detected only with their bases, which come from *adenine*, *guanine*, *cytosine*, and *thymine*, abbreviated A, G, C, and T, respectively. A *DNA strand* is essentially a sequence of four types of nucleotides detected by one of four bases they contain [28]. *DNA-based computing* [24], or more generally *molecular computing*, is a computational paradigm that uses DNA molecules as information storage media. The techniques of molecular biology, such as *polymerase chain reaction* (PCR), *gel electrophoresis*, and *enzymatic reactions*, can be used as computational operators for copying, sorting, and splitting/concatenating the information in the DNA molecules, respectively [1].

Through the progress in molecular biology, it is now possible to produce about $10^{18}$ DNA strands contained in a test tube [28]. We can use each DNA strand to represent a piece of information. Primitive biological operations can be employed to operate $10^{18}$ pieces of information simultaneously. It has the same computing power as $10^{18}$ processors running in parallel. Accordingly, DNA-based computing can provide a huge parallelism for dealing with the intractable problems in the real world.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Feynman [14] first proposed DNA-based computation in 1961, but his idea was not implemented by experiment for a few decades. Adleman [1] was the first researcher who succeeded in solving the Hamiltonian path problem by properly manipulating DNA strands as the input instance of the problem in a test tube. Next, Lipton [21] demonstrated the power of DNA-based computing using the Adleman techniques to solve the satisfiability problem. After that, many researchers studied computational hard problems using the Adleman–Lipton model [17, 23, 29, 22, 13, 27, 3, 4, 5, 6, 7, 16, 8]. A few years later, Roweis *et al.* [26] proposed the concept of *sticker* for enhancing the Adleman–Lipton model. Since then, several nondeterministic polynomial (NP)-complete problems were solved using the Adleman–Lipton model with stickers [26, 25, 9, 10, 18, 11, 19, 20]. The NP-complete problems solved using the Adleman–Lipton model further include the traveling salesman problem, the dominating set problem, the vertex cover problem, the maximum clique problem, the maximum independent set problem, the three-dimensional matching problem, the knapsack problem, the set packing problem, the subset sum problem, the set cover problem, and so on.

This chapter describes a DNA-based graph encoding scheme that can be used to solve the subgraph isomorphism problem and related problems [20] in the Adleman–Lipton model with stickers using a polynomial number of basic biological operations. Theoretically, the subgraph isomorphism problem, which is known to be an NP-complete problem [15], is a common generalization of many important graph problems including finding Hamiltonian paths, cliques, matchings, girth, and shortest paths. Variations of the subgraph isomorphism problem also have been used to model varied practical problems such as molecular structure comparison, integrated circuit testing, and microprogrammed controller optimization [12].

This chapter is organized as follows. In Section 9.2, we introduce the graph isomorphism problem, the subgraph isomorphism problem, and the maximum common subgraph problem. In Section 9.3, we detail the Adleman–Lipton model with stickers. In Section 9.4, we present DNA-based algorithms to generate the solution space for the graph problems. In Section 9.5, complete algorithms for the subgraph isomorphism problem, graph isomorphism problem, and the maximum common subgraph problem are presented, respectively. In Section 9.6, we provide DNA sequences for experiments using our algorithms. Conclusions are given in Section 9.7.

## 9.2 DEFINITIONS OF SUBGRAPH ISOMORPHISM PROBLEM AND RELATED PROBLEMS

Without loss of generality, we only consider undirected simple graphs (i.e., graphs without loops and multiple edges) throughout this chapter. We denote the vertex and edge sets of a graph $G$ by $V(G)$ and $E(G)$, respectively. The numbers of edges in $G$ is called the *size* of $G$. A graph $G'$ is a *subgraph* of $G$, denoted by $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. Given two graphs $G$ and $H$, we say “$G$ is

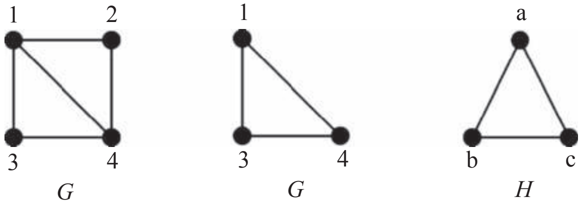


**Figure 9.1** An example of the subgraph isomorphism problem in which $G'$ is the subgraph of $G$ and $G'$ is isomorphic to $H$.

isomorphic to $H$," written $G \cong H$, if a one-to-one and onto function exists, then it is called an *isomorphism* from $G$ to $H$, $\phi : V(G) \rightarrow V(H)$ so that $(u, v) \in E(G)$ if and only if $(\phi(u), \phi(v)) \in E(H)$. The definitions of the *graph isomorphism problem*, the *subgraph isomorphism problem*, and the *maximum common subgraph problem* are described as follows.

**Definitio 9.1** *Given two graphs G and H, the* graph isomorphism problem *is defined to determine whether G is isomorphic to H.*

Notice that the graph isomorphism problem is currently still unknown to be in P or in NP-complete [15].

**Definitio 9.2** *Given two graphs G and H, respectively, called source graph and target graph, the* subgraph isomorphism problem *is defined to determine whether there is a subgraph of the source graph G isomorphic to the target graph H.*

## ■ EXAMPLE 9.1

In Figure 9.1, $G$ is a source graph and $H$ is a target graph. $G$ contains a subgraph $G'$ isomorphic to $H$ with an isomorphism $\phi$: $V(G') \rightarrow V(H)$ defined by $\phi(1) = a$, $\phi(3) = b$, and $\phi(4) = c$.

**Definitio 9.3** *Given two graphs G and H, the* maximum common subgraph problem *is defined to find a subgraph of G with maximum size that is isomorphic to a subgraph of H.*

## ■ EXAMPLE 9.2

In Figure 9.2, $G'$ and $H'$ are a pair of maximum isomorphic subgraphs of graphs $G$ and $H$, respectively, with an isomorphism $\phi$: $V(G') \rightarrow V(H')$ defined by $\phi(1) = a$, $\phi(2) = b$, $\phi(3) = c$, and $\phi(4) = d$.

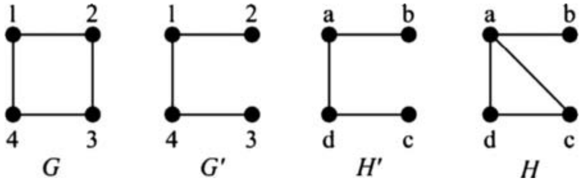


**Figure 9.2** An example of the maximum common subgraph problem in which $G'$ and $H'$ are a pair of maximum isomorphic subgraphs of graphs $G$ and $H$, respectively.

## 9.3 DNA COMPUTING MODELS

In this section, we introduce the model we adopt—the Adleman–Lipton model with stickers.

### 9.3.1 The Stickers

The *stickers model* [2, 24, 26] employs two basic groups of single-stranded DNA molecules in its representation of a bit string. In short, the model involves a long single memory strand and several sticker strands or stickers as indicated (see Figure 9.3). A *memory strand* is a single-stranded DNA with $n$ bases. It is divided into $k$ nonoverlapping substrands, each of which has $m$ bases (*i.e.*, $n = km$). Each *sticker* has $m$ bases and is complementary to exactly one of the $k$ substrands in the memory strand. During a course of computation, each substrand is identified as a Boolean variable and is considered "on" (1) or "off" (0) as to whether its corresponding sticker is annealed or not: If a sticker is annealed to its matching region on a given memory strand, then the bit corresponding to that particular region is on for that strand. If no sticker is annealed to a region, then that region's bit is off. Each memory strand along with its annealed stickers (if any) represents one bit string.

**Stickers**

5'-$\boxed{\text{ATTAATTC}}$-3'  5'-$\boxed{\text{CATTCCAA}}$-3'  5'-$\boxed{\text{AACAACTC}}$-3'  5'-$\boxed{\text{TAACACAT}}$-3'

**Memory strand**

3'-TAATTAAGGTATTGGTGTAAGGTTGGGAAATATTGTTGAGATTAGGGAATTGTGTAAAGGAAGT-5'

**Memory complex**

```
[ATTAATTC]        [CATTCCAA]        [AACAACTC]        [TAACACAT]
TAATTAAGGTATTGGTGTAAGGTTGGGAAATATTGTTGAGATTAGGGAATTGTGTAAAGGAAGT
    1       0       1       0       1       0       1       0
```

**Figure 9.3** Illustration of the stickers model that encodes 10101010. Note that the memory strand consists of eight eight-basis-long subsections. Each subsection is defined as 1 by annealing a sticker and defined as 0 by annealing no sticker.

A *memory complex* is the term defined as a memory strand in which part of the substrands is annealed by the matching stickers, such that the computational information can be carried in a binary format along the memory complex.

### 9.3.2 The Adleman–Lipton Model

A *set* is a group of distinct unordered objects. Different from a set, a *multiset* is a group of unordered objects that allows an object to occur more than once. In the Adleman–Lipton model [1], a tube is a multiset of DNA strands over an alphabet set {A, G, C, T}. Given tubes, one can perform the following operations:

1. Extract($T$, $S$): Given a tube $T$ and a short single strand of DNA, say $S$, the operation produces two tubes $(T, S)^+$ and $(T, S)^-$, where $(T, S)^+$ consists of all molecules of DNA in $T$ such that each contains $S$ as a substrand, and $(T, S)^-$ consists of all molecules of DNA in $T$ that do not contain $S$. Finally, the tube $T$ becomes empty.
2. Merge($T_1$, $T_2$): Given tubes $T_1$ and $T_2$, the operation is to pour two tubes into one, without any change in the individual strands.
3. Detect($T$): Given a tube $T$, the operation returns "yes" if tube $T$ contains at least one DNA molecule. Otherwise, it returns "no."
4. Amplify($T$, $T_1$, $T_2$): Given a tube $T$, the operation produces two tubes $T_1$ and $T_2$ such that $T_1$ and $T_2$ contain the "original copy of those molecules in $T$," and then tube $T$ becomes empty after this operation.
5. Read($T$): Given a tube $T$, the operation is to describe a single molecule contained in $T$. Moreover, the operation can give an explicit description of exactly one of them even if $T$ contains many different molecules, each encoding a different set of bases.
6. Append($T$, $S$): Given a tube $T$ and a short DNA strand $S$, the operation appends $S$ onto the end of every strand in $T$.

## 9.4 THE STICKER-BASED SOLUTION SPACE

Given the set $\{1, 2, \ldots, n\}$, a linear arrangement of these $n$ integers is called an *n-permutation*. The set of all $n$-permutations is called the *n-permutation set*, denoted by $\mathcal{P}(n)$. In the remainder of this chapter, we label the vertices of any $n$-vertex graph as integers from 1 to $n$. The first step of our strategy is to generate the solution space using DNA strands in which each DNA strand contains the graph information including an $n$-permutation of the vertex labels and its corresponding adjacency relation. Next, we check whether a desired solution exists in the solution space for decision.

To indicate the position of any item in an $n$-permutation, we attach the position symbol in front of each item of the $n$-permutation. For example, 3-permutation 213 is represented by "$p_1 2 p_2 1 p_3 3$," where symbol $p_i$ denotes position $i$.

In our algorithm, integers and positions are represented by their binary representations using stickers. We define seven symbols represented by 15-base stickers to encode the information into DNA strands:

- $x^0$ ($x^1$): binary bit 0 (1) for representing integers of permutations
- $p^0$ ($p^1$): binary bit 0 (1) for representing positions
- $y^0$ ($y^1$): binary bit 0 (1) for representing adjacency relation of two vertices
- $\|$: separator symbol

For example, 3-permutation 213 is encoded as "$p^0p^1 \parallel x^1x^0 \parallel p^1p^0 \parallel x^0x^1 \parallel p^1p^1 \parallel x^1x^1 \parallel$." Such representation is called the *DNA-representation* of 213. For short, we use $B_I(i)$ and $B_P(p)$ to denote the DNA representation of integer $i$ and position $p$, respectively.

### 9.4.1 Using Stickers for Generating the Permutation Set

With repetition of elements allowed, a sequence with $n$ elements of $\{1, 2, \ldots, n\}$ is called an *$n$-sequence*. The set of all possible $n$-sequences is called the *$n$-sequence set*, denoted by $\mathcal{S}(n)$. For example, $\mathcal{S}(2) = \{11, 12, 21, 22\}$. Notice that an $n$-permutation is also an $n$-sequence. Thus, the $n$-permutation set is a subset of the $n$-sequence set, that is, $\mathcal{P}(n) \subset \mathcal{S}(n)$. There is a simple recursive way to generate $\mathcal{S}(n)$ by appending all integers one by one and digit by digit, which is presented in Algorithm 9.1. In this algorithm, we insert the position symbols to the $n$-sequences for efficient indexing in the successive algorithms.

**Algorithm 9.1**

```
Sequence_Generation($n$) {
  Input: A nonnegative integer $n$.

  Output A tube $T_\sigma$ containing the $n$-sequence set.

  Initially, tube $T_\sigma$ contains only one DNA strand that is a pure
    strand without any encoding.
  for $i = 1$ to $n$
    Append($T_\sigma$, "$B_P(i) \parallel$ ")
    for $j = 1$ to $\lceil \log_2(n+1) \rceil$
      Amplify($T_\sigma, T_0, T_1$)
      Append($T_0$, "$x^0$")
      Append($T_1$, "$x^1$")
      $T_\sigma$ := Merge($T_0, T_1$)
    end for
    Append($T_\sigma$, " $\parallel$ ")
  end for
}
```

**Lemma 9.1** *Algorithm 9.1 correctly generates $\mathcal{S}(n)$ and takes $\Theta(n \log n)$* Append, Amplify, *and* Merge *operations.*

*Proof:* Without loss of generality, we assume that $n$ is an integer of the power of 2. Proved by induction on $i$, when $i = 1$, it is easy to verify that $T_\sigma$ contains DNA strands of "$B_P(1) \parallel B_I(x) \parallel$" for $0 \le x \le 2^{\lceil \log_2(n+1) \rceil}$. Assume that $T_\sigma$ contains $\mathcal{S}(n-1)$ when $i = n-1$. When $i = n$, "$B_P(i) \parallel$" is first appended to all $(n-1)$ sequences. In the $j$ for-loop, we append each possible integer, with binary bit length $\lceil \log_2(n+1) \rceil$, to all $(n-1)$ sequences in $\mathcal{S}(n-1)$. Thus, $\mathcal{S}(n)$ is generated.

The number of operations can be obtained directly from the algorithm. ■

In Algorithm 9.2, the $n$-permutation set is generated by removing the $n$-sequences that are not $n$-permutations from the given $n$-sequence set.

**Algorithm 9.2**

```
Permutation_Generation($n$) {
  Input: A nonnegative integer $n$.

  Output: A tube $T_\pi$ containing the $n$-permutation set.

  1:Call Sequence_Generation($n$) to generate a tube $T_\pi$ containing the
    $n$-sequence set.
  2: for $i = 1$ to $n$
  3:   Extract($T_\pi$, "$B_I(i)$")
  4:   $T_\pi := (T_\pi, $"$B_I(i)$"$)^+$
  5: end for
}
```

**Lemma 9.2** *Algorithm 9.2 correctly generates $\mathcal{P}(n)$ and takes $\Theta(n)$* Extract *operations.*

*Proof:* Because an $n$-permutation is also an $n$-sequence, and an $n$-sequence contains all elements in $\{1, 2, \ldots, n\}$ it must be an $n$-permutation, the claim follows. The number of operations can be obtained directly from the algorithm. ■

### 9.4.2 Using Stickers for Generating the Solution Space

After generating the $n$-permutation set of the given $n$-vertex graph $G$, for each permutation, we encode the adjacency relation of $G$ to be appended to the corresponding DNA strand in parallel, which is presented in Algorithm 9.3. Because the graphs we considered are undirected, the adjacency matrices of the graphs are symmetric. Thus, the entries in the upper triangle of the adjacent matrix of $G$ are enough to represent the adjacency relation of $G$. That is, there are $\frac{n(n-1)}{2}$ 15-base stickers $y_1 y_2 \ldots y_{\frac{n(n-1)}{2}}$

appended to every $n$-permutation $\alpha_1\alpha_2 \ldots \alpha_n$ DNA strand based on the following two rules:

**Rule 9.1** The $\frac{n(n-1)}{2}$ 15-base stickers $y_1, y_2, \ldots, y_{\frac{n(n-1)}{2}}$ represent the adjacency relation of $\frac{n(n-1)}{2}$ pairs of vertices $(\alpha_1, \alpha_2)$, $(\alpha_1, \alpha_3)$, …, $(\alpha_1, \alpha_{n-1})$, $(\alpha_2, \alpha_3)$, $(\alpha_2, \alpha_4)$, …, $(\alpha_2, \alpha_{n-1})$, …, $(\alpha_{n-2}, \alpha_{n-1})$, $(\alpha_{n-2}, \alpha_n)$, and $(\alpha_{n-1}, \alpha_n)$, respectively. For $1 \leq i < j \leq n$, the adjacency relation of $(\alpha_i, \alpha_j)$ is represented by $y_k$, where $k = n(i-1) - \frac{(i)(i-1)}{2} + (j-i)$.

**Rule 9.2** For $k = n(i-1) - \frac{(i)(i-1)}{2} + (j-i)$,

$$y_k = \begin{cases} y^1 & \text{if } \alpha_i \text{ and } \alpha_j \text{ are adjacent} \\ y^0 & \text{if } \alpha_i \text{ and } \alpha_j \text{ are not adjacent.} \end{cases} \tag{9.1}$$

### Algorithm 9.3

```
Solution_Space($G, n$) {
  Input: A graph $G$ and a positive integer $n$ that is the number of
         vertices in $G$.
  Output: A tube $T_{adj}$ contains all $n$-permutations associated with
          the adjacency relation of $G$.

  $T_{adj}$ := Permutation_Generation($n$);
  $k = 0$;
  for $p = 1$ to $n - 1$
    for $q = i + 1$ to $n$
      for $i = 1$ to $n$
        for $j = 1$ to $n$
          if $i \neq j$
            $T_a$ := (Extract($T_{adj}$, "$B_P(p) \parallel B_I(i)$"))$^-$;
            $T_i$ := (Extract($T_{adj}$, "$B_P(p) \parallel B_I(i)$"))$^+$;
            $T_b$ := (Extract($T_i$, "$B_P(q) \parallel B_I(j)$"))$^-$;
            $T_{i,j}$ := (Extract($T_i$, "$B_P(q) \parallel B_I(j)$"))$^+$;
            $k = k + 1$;
            if vertex $i$ is adjacent to vertex $j$ in $G$ then
              Append($T_{i,j}$, "$B_P(k) \parallel y^1 \parallel$");
            else
              Append($T_{i,j}$, "$B_P(k) \parallel y^0 \parallel$");
            end if
              $T_{\text{adj}}$ := Merge($T_{i,j}$, Merge($T_a$, $T_b$));
            end if
          end for
        end for
      end for
    end for
  end for
}
```

**Lemma 9.3** *After executing, Algorithm 9.3 consists of all* n*-permutations associated with the adjacency relation of G, satisfying Rules 1 and 2.*

*Proof:* In the algorithm, we use the indices $p$ and $q$ to represent positions and the indices $i$ and $j$ to represent the labels of vertices. All pairs of vertices described in Rule 9.1 are considered sequentially. From lines 7–10, we extract all DNA strands in which vertex $i$ is located at position $p$ and vertex $j$ is located at position $q$ from $T_{i,j}$. From lines 11–17, if the vertex $i$ is adjacent (respectively, nonadjacent) to the vertex $j$, then sticker representing 1 (respectively, 0) will be appended to all DNA strands in $T_{i,j}$. Therefore, Rule 9.2 also holds. ■

**Lemma 9.4** *Algorithm 9.3 takes* $\Theta(n^4)$ Extract, Append, *and* Merge *operations.*

*Proof:* It follows directly from the algorithm. ■

## 9.5 ALGORITHMS FOR SOLVING PROBLEMS

In this section, we present DNA-based algorithms for solving the subgraph isomorphism problem, the graph isomorphism problem, and the maximum common subgraph problem based on biological operations in the Adleman–Lipton model and the solution space of stickers in the sticker-based model.

### 9.5.1 Solving the Subgraph Isomorphism Problem

After executing Algorithm Solution_Space($G$, $n$), each DNA strand in the result tube of the solution space consists of two parts, an *n-permutation string* $P$ and the *adjacency string* $Y$ with respect to $P$. We call $PY$ the *string representation* of $G$. For convenience, we write string representations without the position and separator symbols.

**Lemma 9.5** *Given two n-vertex graphs G and H, if a pair of string representations exists* $PY$ *and* $P'Y'$ *of G and H, respectively, such that* $Y = Y'$, *then G is isomorphic to H.*

*Proof:* Suppose $P = \alpha_1\alpha_2 \ldots \alpha_n$, $P' = \alpha'_1\alpha'_2 \ldots \alpha'_n$, $Y = y_e y_2 \ \ldots \ y_{\frac{n(n-1)}{2}}$, and $Y' = y'_1 y'_2 \ \ldots \ y'_{\frac{n(n-1)}{2}}$. Recall that each adjacency bit $y_k$ is decided by the adjacency relation of the pair of vertices $(\alpha_i, \alpha_j)$ at determined positions $i$ and $j$ such that $k = n(i-1) - \frac{(i)(i-1)}{2} + (j-i)$. Because $Y = Y'$, we have that $y_i = y'_i$ for $1 \leq i \leq \frac{n(n-1)}{2}$. Thus, we can obtain an isomorphism $\phi$ from $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ to $\{\alpha'_1, \alpha'_2, \ldots, \alpha'_n\}$ such that $\phi(\alpha_p) = \alpha'_p$ for $1 \leq p \leq n$ and $(\alpha_i, \alpha_j) \in E(G)$ (*i.e.*, $y_k = 1$) if and only if $(\alpha'_i, \alpha'_j) \in E(H)$ (*i.e.*, $y'_k = 1$) for $1 \leq i < j \leq n$. By definition, $G$ is isomorphic to $H$. ■

Let $s = b_1 b_2 \ldots b_n$ and $s' = b'_1 b'_2 \ldots b'_n$ be two binary strings of the same length. We say "$s$ is an *overlay string* of $s'$" if the following property is satisfied: for $1 \leq i \leq$

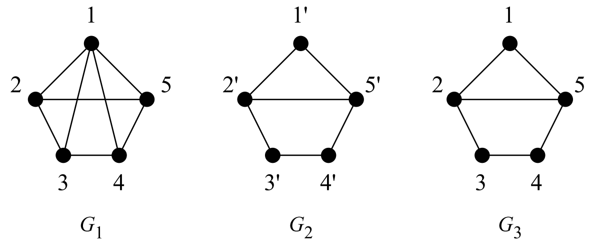


**Figure 9.4** An example of the subgraph isomorphism problem in which $G_3$ is the subgraph of $G_1$, and $G_3$ is isomorphic to $G_2$.

$n$, if $b'_i = 1$, then $b_i = 1$. For example, 1011 is an overlay string of 1010, but 1011 is not an overlay string of 1110.

**Lemma 9.6** *Given two $n$-vertex graphs $G$ and $H$, if a pair of string representations exists $PY$ and $P'Y'$ of $G$ and $H$, respectively, such that $Y$ is an overlay string of $Y'$, then $G$ contains a subgraph isomorphic to $H$.*

*Proof:* Because each bit in the adjacency string represents the existence of an edge, it is clear that any subgraph $G'$ of $G$ with $V(G') = V(G)$ and $E(G') \subseteq E(G)$ must have a string representation $P^-Y^-$ such that $P^- = P$ and $Y$ is an overlay string of $Y^-$. Because $Y$ is an overlay string of $Y'$, a subgraph $G'$ of $G$ exists with string representation $PY'$. By Lemma 9.5, $G'$ is isomorphic to $H$. ■

## ■ EXAMPLE 9.3

In Figure 9.4, $G_1$ and $G_2$ are five-vertex graphs. The string representations of $G_1$ and $G_2$ are $PY$ = 12345 1111101101 and $P'Y'$ = 1′2′3′4′5′ 1001101101, respectively. Obviously, $Y$ is an overlay string of $Y'$. We can find a subgraph $G_3$ of $G_1$ constructed by removing the edges (1,3) and (1,4) from $G_1$ with the string representations $PY'$ = 12345 1001101101. Because the adjacency strings of $G_2$ and $G_3$ are the same, it follows that $G_2 \cong G_3$ by Lemma 9.5. An isomorphism exists, $\phi: V(G_2) \to V(G_3)$ such that $\phi(1') = 1, \phi(2') = 2, \phi(3') = 3, \phi(4') = 4$, and $\phi(5') = 5$.

A vertex with degree 0 is called an *isolated vertex*. Given an $m$-vertex graph $H$ and an integer $n > m$, we define the *n-extension graph* of $H$ as the graph $H_n^+$ such that $E(H_n^+) = E(H)$ and $V(H_n^+) = V(H) \cup V_e$, where $V_e$ is a set of isolated vertices, called *pseudovertices*, labeled from $m + 1$ to $n$.

**Lemma 9.7** *Given an $n$-vertex graph $G$ and an $m$-vertex graph $H$ for $m < n$, if a pair of string representations exists $PY$ and $P^+Y^+$ of $G$ and $H_n^+$ (the n-extension*

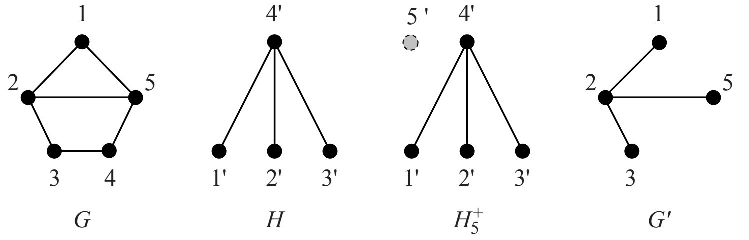


**Figure 9.5** $G$ and $H$ are five-vertex and four-vertex graphs, respectively. The graph $G'$ is a subgraph of $G$, which is isomorphic to $H$.

*graph of $H$), respectively, such that $Y$ is an overlay string of $Y^+$, then $G$ contains a subgraph isomorphic to $H$.*

*Proof:* By Lemma 9.6, we have that $G$ contains a subgraph isomorphic to $H_n^+$. Suppose $G'$ is a subgraph of $G$, which is isomorphic to $H_n^+$. Then an isomorphism $\phi$ exists in which $V(G') \to V(H_n^+)$. Let $V_e$ be the set of pseudovertices in $H_n^+$ and $V_e'$ be the vertex subset of $V(G')$ that maps to $V_e$ by $\phi$. Because the pseudovertices in $H_n^+$ are isolated vertices, it follows that the vertices in $V_e'$ are also isolated vertices in $G'$. Then we can construct graph $G''$ by removing the vertices in $V_e'$ from $G'$. That is, $V(G'') = V(G') \setminus V_e'$ and $E(G'') = E(G')$. $G''$ is obviously also a subgraph of $G$. Let us consider a mapping $\phi'$ in which $(V(G') \setminus V_e') \to (V(H_n^+) \setminus V_e)$ (*i.e.*, $\phi' : V(G'') \to V(H)$). Because $E(G') = E(G'')$ and $E(H_n^+) = E(H)$, we have that $\phi'$ is an isomorphism from $G''$ to $H$. Thus, the claim follows. ■

## ■ EXAMPLE 9.4

In Figure 9.5, $G$ and $H$ are five-vertex and four-vertex graphs, respectively. Let $P'Y' = 1'2'3'4'\ 001011$ be a string representation of $H$. Let $H_n^+$ be a 5-extension graph of $H$ by adding one pseudovertex $5'$. The string representation of $H_n^+$ with respect to the five-permutation $1'2'3'4'5'$ is $P^+Y^+ = 1'2'3'4'5'\ 0010010100$. Let us consider the string representation $PY = 35124\ 0011111100$ of $G$ with respect to the five-permutation 35124. Clearly, $Y$ is an overlay string of $Y^+$. By Lemma 9.7, $G$ has a subgraph isomorphic to $H$. For example, $G'$ is a subgraph of $G$ isomorphic to $H$.

Let $G$ be the source graph with $n$ vertices and $H$ be the target graph with $m$ vertices. Notice that $n \geq m$. We first generate all $n$-permutations associated with the adjacency relations of $G$ by Algorithm 9.3. Next, we construct the $n$-extension graph $H_n^+$ of $H$. Finally, we check in parallel whether an $n$-permutation exists whose corresponding adjacency string is an overlay string of the adjacency string of $H_n^+$.

## Algorithm 9.4

```
Solving_Subgraph_Isomorphism($G, H, n, m$) {
  Input: (1) $G$ is the source graph and $H$ is the target graph.
         (2) $n$ is the number of vertices of $G$.
         (3) $m$ is the number of vertices of $H$.
  Output: All isomorphisms from $G'$ to $H$ for all subgraphs $G'$ of $G$
          that are isomorphic to $H$. If no such subgraph exists,
          then it outputs the message that "$G$ contains no subgraph
          isomorphic to $H$."

   1:Call Solution_Space($G, n$) to generate $T_{adj}$;
   2:$H^+$ is the $n$-extension graph of $H$ by adding $n - m$ pseudovertices
     labeled from $m + 1$ to $n$;
   3: for $i = 1$ to $m - 1$
   4:   for $j = i + 1$ to $m$
   5:     if vertex $i$ and vertex $j$ are adjacent in $H^+$
   6:       $k = n(i - 1) - \frac{(i)(i-1)}{2} + (j - i)$;
   7:       Extract($T_{\text{adj}}$, "$B_P(k) \parallel y^1$");
   8:       $T_{adj} := (T_{\text{adj}}$, "$B_P(k) \parallel y^1$")+;
   9:     end if
  10:   end for
  11: end for
  12: if Detect($T_{\text{adj}}$) ="$yes$"
  13:   Read($T_{\text{adj}}$);
  14: else
  15:   Output "$G$ contains no subgraph isomorphic to $H$";
  16: end if
}
```

## ■ EXAMPLE 9.5

Let us consider two graphs $G$ and $H$ shown in Figure 9.6. The adjacency string of $H_4^+ = 110100$. After executing the algorithm Solving_Subgraph _Isomorphism($G$, $H$, 4, 3), tube $T_{\text{adj}} = \{1234\ 110111,\ 1324\ 110111,\ 2134\ 111101,\ 2314\ 111110,\ 2341\ 111110,\ 2431\ 111101,\ 3124\ 111101,\ 3214\ 111110,\ 3241\ 111110,\ 3421\ 111101,\ 4231\ 110111,\ 4321\ 110111\}$. By applying Read operations in line 13, we obtain 12 DNA strands. This implies that $G$

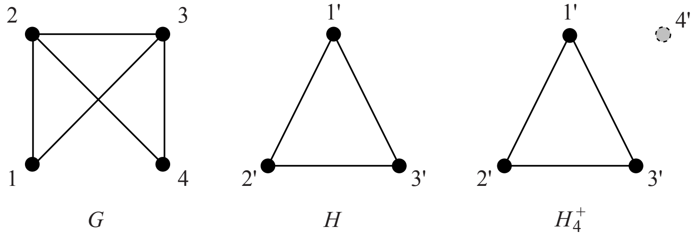


**Figure 9.6** $G$ and $H$ are four-vertex and three-vertex graphs, respectively. $H_4^+$ is a 4-extension graph of $H$.

**Table 9.1 Twelve isomorphisms are obtained from the string representations of strands in $T_{adj}$ and the adjacency string of $H_4^+$**

| $\pi$ | $V(G') \rightarrow V(H)$ |
|---|---|
| $\pi_1$ : | $\pi_1(1) = 1',\ \pi_1(2) = 2',\ \pi_1(3) = 3'$ |
| $\pi_2$ : | $\pi_2(1) = 1',\ \pi_2(3) = 2',\ \pi_2(2) = 3'$ |
| $\pi_3$ : | $\pi_3(2) = 1',\ \pi_3(1) = 2',\ \pi_3(3) = 3'$ |
| $\pi_4$ : | $\pi_4(2) = 1',\ \pi_4(3) = 2',\ \pi_4(1) = 3'$ |
| $\pi_5$ : | $\pi_5(2) = 1',\ \pi_5(3) = 2',\ \pi_5(4) = 3'$ |
| $\pi_6$ : | $\pi_6(2) = 1',\ \pi_6(4) = 2',\ \pi_6(3) = 3'$ |
| $\pi_7$ : | $\pi_7(3) = 1',\ \pi_7(1) = 2',\ \pi_7(2) = 3'$ |
| $\pi_8$ : | $\pi_8(3) = 1',\ \pi_8(2) = 2',\ \pi_8(1) = 3'$ |
| $\pi_9$ : | $\pi_9(3) = 1',\ \pi_9(2) = 2',\ \pi_9(4) = 3'$ |
| $\pi_{10}$ : | $\pi_{10}(3) = 1', \pi_{10}(4) = 2', \pi_{10}(2) = 3'$ |
| $\pi_{11}$ : | $\pi_{11}(4) = 1', \pi_{11}(2) = 2', \pi_{11}(3) = 3'$ |
| $\pi_{12}$ : | $\pi_{12}(4) = 1', \pi_{12}(3) = 2', \pi_{12}(2) = 3'$ |

contains subgraphs $G'$ that are isomorphic to $H$. Moreover, 12 isomorphisms $\pi_1, \pi_2, \ldots, \pi_{12} : V(G') \rightarrow V(H)$ are obtained, and they are shown in Table 9.1.

**Theorem 9.1** *Algorithm 9.4 solves the subgraph isomorphism problem and takes $\Theta(|E(H)|)$ Extract operations, excluding the operations for generating the permutations.*

*Proof:* The correctness of this algorithm follows from Lemma 9.5. ∎

## 9.5.2 Solving the Graph Isomorphism Problem

Because a graph is a subgraph of itself, we easily can revise Algorithm 9.4 to the algorithm for solving the graph isomorphism problem as follows:

### Algorithm 9.5

```
Solving_Graph_Isomorphism($G$, $H, n, m$) {
  Input: (1) $G$ and $H$ are two graphs.
         (2) $n$ is the number of vertices of $G$.
         (3) $m$ is the number of vertices of $H$.
  Output: All isomorphisms from $G$ to $H$ if $G$ is isomorphic
          to $H$. If $G$ is not isomorphic to $H$, then it outputs the
          message that "$G$ is not isomorphic to $H$."

  if $n \neq m$;
    Output "$G$ is not isomorphic to $H$";
  else
    Call Solution_Space($G, n$) to generate $T_{adj}$;
    for $i = 1$ to $n - 1$
      for $j = i + 1$ to $n$
```

```
            if vertex $i$ and vertex $j$ are adjacent in $H$
              $k = n(i-1) - \frac{(i)(i-1)}{2} + (j-i)$;
              Extract($T_{\mathrm{adj}}$, "$B_P(k) \| y^1$");
              $T_{\mathrm{adj}} := (T_{\mathrm{adj}}$, "$B_P(k) \| y^1$")+;
            end if
          end for
        end for
        if Detect($T_{\mathrm{adj}}$) ="yes"
          Read($T_{\mathrm{adj}}$);
        else
          Output "$G$ is not isomorphic to $H$";
        end if
      end if
}
```

**Theorem 9.2** *Algorithm 9.5 solves the graph isomorphism problem and takes $\Theta(|E(H)|)$ Extract operations, excluding the operations for generating the permutations.*

*Proof:* The correctness of this algorithm follows from Lemma 9.7. ■

### 9.5.3 Solving the Maximum Common Subgraph Problem

Let $s = b_1 b_2 \ldots b_n$ and $s' = b'_1 b'_2 \ldots b'_n$ be two binary strings of the same length. We define the *match number* of $s$ and $s'$ as the number of *matched bits* so that $b_i = b'_i = 1$ for $1 \leq i \leq n$. For example, the match number of $\underline{1}0\underline{1}100$ and $\underline{1}0\underline{1}010$ equals two and the matched bits are underlined.

**Lemma 9.8** *Let $G$ and $H$ be two $n$-vertex graphs, and $PY$ and $P'Y'$ be two string representations of $G$ and $H$, respectively. Suppose that the match number of $Y$ and $Y'$ equals $t$. Then $G$ and $H$ have a common subgraph of size $t$.*

*Proof:* Let $Y_M$ be the string obtained by the Boolean "and" operation on $Y$ and $Y'$ with match number $t$. Obviously, $Y$ and $Y'$ are both overlay strings of $Y_M$. Thus, $PY_M$ and $P'Y_M$ are two string representations of subgraphs of $G$ and $H$, respectively. By Lemma 9.5, we have that these two subgraphs are isomorphic because they have the same adjacency string. Because $Y_M$ has $t$ adjacency bits of 1, the size of the common subgraph is equal to $t$. ■

According to Lemma 9.8, we propose an edge-based algorithm to solve the maximum common subgraph problem. Let $G$ and $H$ be two considered graphs with $n$ and $m$ vertices, respectively. Without loss of generality, we assume that $n \geq m$. In the algorithm, there are $|E(H)|$ stages. In each stage, we classify the permutation strands of $G$ into a set of tubes $T_i$ by checking the adjacency bit corresponding to an edge of $H$, where tube $T_i$ carries the permutation strands with at least $i$ matched adjacency bits. After all adjacency bits corresponding to $E(H)$ are checked, the tube $T_i$ with maximum index carries all permutation strands with maximum matched adjacency bits. Then we can obtain a subgraph and an isomorphism by the positions appended after the adjacency string of each strand in the result tube.

### Algorithm 9.6

```
Solving_Maximum_Common_Subgraph($G$, $H, n, m$) {
    Input: $G$ and $H$ are two graph with $n$ and $m$ vertices, respectively,
           where $n \geq m$.
    Output: All maximum subgraphs of $H$ that are isomorphic to some
           subgraphs of $G$.

    Initial the tube index $t = 0$;
    Call Solution_Space($G, n$) to generate $T_0$;
    $H_n^+$ is the $n$-extension graph of $H$ by adding $n - m$ pseudovertices
      labeled from $m + 1$ to $n$
    for $i = 1$ to $m - 1$
      for $j = i + 1$ to $m$
        if vertex $i$ and vertex $j$ are adjacent in $H^+$
          $k = n(i-1) - \frac{(i)(i-1)}{2} + (j - i)$;
          Extract($T_t$, "$B_P(k) \parallel y^1$");
          if Detect(($T_t$, "$B_P(k) \parallel y^1$")+) = "yes"
            Take a new tube $T_{t+1} := (T_t$, "$B_P(k) \parallel y^1$")+;
            Append($T_{t+1}$, "$B_P(k) \parallel$ ");
          end if
          for $p = t - 1$ down to 0
            Extract($T_p$, "$B_P(k) \parallel y^1$");
            $T_{p+1} := (T_p$, "$B_P(k) \parallel y^1$")+;
            Append($T_{p+1}$, "$B_P(k) \parallel$ ");
            $T_p := (T_p$, "$B_P(k) \parallel y^1$")−;
          end for
          $t$ = the maximum index of the tubes;
        end if
      end for
    end for
    Read($T_t$);
}
```

### ■ EXAMPLE 9.6

Let us consider two graphs $G$ and $H$ shown in Figure 9.7. The adjacency string of $H_4^+ = 110100$. After executing the Algorithm 9.6, tube $T_2 = \{$ 1234 100101 $B_P(1)$ $B_P(4)$, 1324 010110 $B_P(2)$ $B_P(4)$, 2134 110001 $B_P(1)$ $B_P(2)$, 2314 110010 $B_P(1)$ $B_P(2)$, 2341 101100 $B_P(1)$ $B_P(4)$, 2431 011100 $B_P(2)$ $B_P(4)$, 3124 011100 $B_P(2)$ $B_P(4)$, 3214 101100 $B_P(1)$ $B_P(4)$, 3241 110010 $B_P(1)$ $B_P(2)$, 3421 110001 $B_P(1)$ $B_P(2)$, 4231 010110 $B_P(2)$ $B_P(4)$, 4321 100101 $B_P(1)$ $B_P(4)$ $\}$, tube $T_1 = \{$ 1243 100011 $B_P(1)$, 1342 001110 $B_P(4)$, 1423 010011 $B_P(2)$, 1432 001101 $B_P(4)$, 2143 101001 $B_P(1)$, 2413 011010 $B_P(2)$, 3142 011010 $B_P(2)$, 3412 101001 $B_P(1)$, 4123 001101 $B_P(4)$, 4132 010011 $B_P(2)$, 4213 001110 $B_P(4)$, 4312 100011 $B_P(1)$ $\}$, and tube $T_0 = \emptyset$.

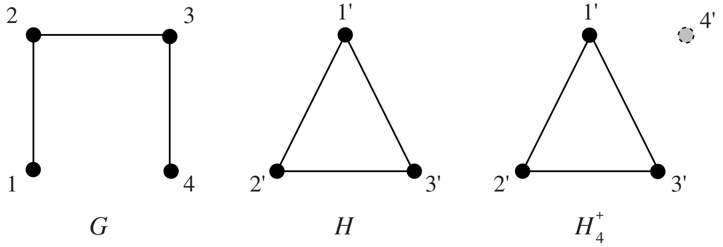


**Figure 9.7** *G* and *H* are four-vertex and three-vertex graphs, respectively. $H_4^+$ is a four-extension graph of *H*.

By applying Read operations on the strands in $T_2$, we obtain 12 isomorphisms $\pi_1, \pi_2, \ldots, \pi_{12}; V(G') \rightarrow V(H')$ are obtained, where $G'$ and $H'$ are subgraphs of $G$ and $H$, respectively, and they are shown in Table 9.2.

**Theorem 9.3** *Algorithm 9.6 solves the maximum common subgraph problem and takes* $\Theta(|E(H)|^2)$ *Extract and Append operations, excluding the operations for generating the permutations.*

*Proof:* Suppose $H'$ is a maximum common subgraph of $H$ with $G$. It follows that the $n$-extension graph of $H'$, $H_n^{+'}$ is also a subgraph of the $n$-extension graph of $H$, $H_n^+$. Assume that $P_hY_h$, and $P_hY'_h$ are the string representations of $H_n^+$ and $H_n^{+'}$, respectively. By Lemma 9.7, we have that $Y_h$ is an overlay string of $Y'_h$. Suppose $G'$ is a subgraph of $G$ isomorphic to $H'$. Because $G'$ is a subgraph of $G$, the $n$-extension graph of $G'$,$G_n^{+'}$ is also a subgraph of $G$. Assume that $P_gY_g$ and $P_gY'_g$ are the string representations of $G$ and $G_n^{+'}$, respectively. By Lemma 9.7, we have that $Y_g$ is also

**Table 9.2 Twelve isomorphisms are obtained from the string representations of strands in $T_2$ and the adjacency string of $H_4^+$**

| $\pi$ | $V(G') \rightarrow V(H')$ |
|---|---|
| $\pi_1$ : | $\pi_1(1) = 1',\ \pi_1(2) = 2',\ \pi_1(3) = 3'$ |
| $\pi_2$ : | $\pi_2(1) = 1',\ \pi_2(3) = 2',\ \pi_2(2) = 3'$ |
| $\pi_3$ : | $\pi_3(2) = 1',\ \pi_3(1) = 2',\ \pi_3(3) = 3'$ |
| $\pi_4$ : | $\pi_4(2) = 1',\ \pi_4(3) = 2',\ \pi_4(1) = 3'$ |
| $\pi_5$ : | $\pi_5(2) = 1',\ \pi_5(3) = 2',\ \pi_5(4) = 3'$ |
| $\pi_6$ : | $\pi_6(2) = 1',\ \pi_6(4) = 2',\ \pi_6(3) = 3'$ |
| $\pi_7$ : | $\pi_7(3) = 1',\ \pi_7(1) = 2',\ \pi_7(2) = 3'$ |
| $\pi_8$ : | $\pi_8(3) = 1',\ \pi_8(2) = 2',\ \pi_8(1) = 3'$ |
| $\pi_9$ : | $\pi_9(3) = 1',\ \pi_9(2) = 2',\ \pi_9(4) = 3'$ |
| $\pi_{10}$ : | $\pi_{10}(3) = 1', \pi_{10}(4) = 2', \pi_{10}(2) = 3'$ |
| $\pi_{11}$ : | $\pi_{11}(4) = 1', \pi_{11}(2) = 2', \pi_{11}(3) = 3'$ |
| $\pi_{12}$ : | $\pi_{12}(4) = 1', \pi_{12}(3) = 2', \pi_{12}(2) = 3'$ |

an overlay string of $Y'_g$. Because $G'$ and $H'$ are isomorphic, it derives that $Y'_g = Y'_h$. Thus, the strand with string representation $P_g Y_g$ will be contained in the tube with maximum index after the execution of Algorithm 9.6. That is, Algorithm 9.6 can find a maximum common subgraph of $G$ and $H$. The number of operations can be obtained directly from the algorithm. ■

## 9.6 EXPERIMENTAL DATA

In our experiment, we used a unique *value sequence*, a 15-base DNA sequence, to implement each symbol of $\{x^0, x^1, y^0, y^1, p^0, p^1, ||\}$ in our algorithms. A *library sequence* is a concatenation of value sequences for representing an instance in the solution space. DNA molecules that carry library sequences are named *library strands*. A *library* is a tube containing library strands, and the *probe* used for separating the library strands have sequences complementary to the value sequences.

In DNA-based computation, there are errors in the separation of the library strands. To make the computation reliable, sequences must be designed to ensure that the following two conditions hold: one is that library strands have little secondary structure that might inhibit intended probe–library hybridization and the other is that the design must exclude sequences that might encourage unintended probe–library hybridization. To help achieve the goals, good sequences were generated to satisfy the following seven constraints defined by Braich *et al.* [2].

1. Library sequences contain only As, Ts, and Cs.
2. All library and probe sequences have no occurrence of five or more consecutive identical nucleotides (*i.e.*, no runs of more than 4 As, 4 Ts, 4 Cs or 4 Gs occur in any library or probe sequences).
3. Every probe sequence has at least four mismatches with all 15-base alignment of any library sequence (except for with its matching value sequence).
4. Every 15-base subsequence of a library sequence has at least four mismatches with all 15-base alignment of itself or any other library sequence.
5. No probe sequence has a run of more than seven matches with any eight base alignment of any library sequence (except for with its matching value sequence).
6. No library sequence has a run of more than seven matches with any eight base alignment of itself or any other library sequence.
7. Every probe sequence has four, five, or six Gs in its sequence.

We used BioPython, a python package for computational molecular biology, to generate good DNA sequences that are suitable for running our algorithms in the laboratory. The package and its documents can be downloaded from the web site http://biopython.org/. Moreover, we built a web server at http://algorithm.csie.ncku.edu.tw:18080/DNA/ to generate DNA sequences satisfying the constraints. In our

**Table 9.3 DNA sequences for solving the graph problems**

| Sticker | DNA sequence | G-C ratio | DeltaS | DeltaH | $T_m$ |
|---|---|---|---|---|---|
| $x^0$ | CCTATACCCATACCC | 53.33 | 313.73 | 108.9 | 38.08 |
| $x^1$ | CCCATATACACCTCA | 46.66 | 313.33 | 108.7 | 37.87 |
| $p^0$ | TTAACATCTCCTATT | 26.66 | 308.63 | 104.8 | 30.80 |
| $p^1$ | CTCCTCCACCCTAAT | 53.33 | 310.13 | 108.6 | 40.45 |
| $y^0$ | CTAAATCCATACCTC | 40.0 | 320.13 | 109.3 | 33.62 |
| $y^1$ | TCTTCCTCTCAAATC | 40.0 | 317.73 | 109.4 | 35.98 |
| $\|\|$ | ATACCACTATACCAA | 33.33 | 307.53 | 105.3 | 33.23 |

system, the seven constraints are optional. After the constraints are selected, the system will generate new DNA sequences that satisfy the selected constraints. The DNA sequences generated for the algorithm using our system are described in Table 9.3. Our system also can compute the G-C ratio, DeltaH, DeltaS, and melting temperature $T_m$. *G-C ratio* is the percentage of G or C in its sequence. *Enthalpy* and *entropy* are two properties for thermodynamics. Enthalpy comes from its greek meaning "heat inside," and entropy is a measure of the disorder of a system. *DeltaH* is defined as the enthalpy change, and *DeltaS* is the entropy change. The *melting temperature* is defined as the temperature at which half of all duplexes are denatured.

## 9.7 CONCLUSION

In this chapter, we have presented DNA-based algorithms for solving the subgraph isomorphism problem, the graph isomorphism problem, and the maximum common subgraph problem based on biological operations in the Adleman–Lipton model and the solution space of stickers in the sticker-based model. Our algorithms provide a DNA-based graph encoding scheme to these graph problems. The algorithms can be performed in a fully automated manner in a laboratory. Furthermore, we have developed a web server to generate good DNA sequences for generating the solution space of the graph problems. It ensures the presented algorithms have a low rate of errors for hybridization.

## REFERENCES

1. L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
2. R.S. Braich, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, M. Leonard, and L.M. Adleman, Solution of a satisfiability problem on a gel-based DNA computer. *Proceedings of the Sixth International Conference on DNA Computation* (*DNA 2000*), *Lecture Notes in Computer Science*, volume 2054, 2001, pp. 27–42.
3. M. Guo, M. Ho, and W.L. Chang. Fast parallel molecular solution to the dominating-set problem on massively parallel bio-computing. *Parallel Comput*, 30(9–10):1109–1125, 2004.

4. M. Ho, W.L. Chang, M. Guo, and L. T. Yang. Fast parallel solution for set-packing and clique problems by DNA-based computing. *IEICE Trans Inf Syst*, E-87D(7):1782–1788, 2004.
5. M. Guo, W.L. Chang, and M. Ho, J. Lu, and J. Cao. Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-based computing. *BioSystems*, 80(1):71–82, 2005.
6. W.L. Chang, M. Guo, and J. Cao. Using Sticker to solve the 3-dimensional matching problem in molecular supercomputers. *IJHPCN*, 1(1/2/3):128–139, 2004.
7. W.L. Chang. Fast parallel DNA-based algorithms for molecular computation: The set-partition problem. *IEEE Trans Nanobiosci*, 6(1):346–353, 2007.
8. W.L. Chang and M. Guo. Solving the set cover problem and the problem of exact 3 cover by 3-sets in the Adleman-Lipton model. *Biosystems*, 72(3):263–275, 2003.
9. W.L. Chang, M. Guo, and M. (Shan-Hui) Ho. Solving the set-splitting problem in sticker-based model and the AdlemanVLipton model. *Future Generation Comput Syst*, 20(5):875–885, 2004.
10. W.L. Chang, M. Guo, and M. (Shan-Hui) Ho. Molecular solutions for the subset-sum problem on DNA-based supercomputing. *Biosystems*, 73(2):117–130, 2004.
11. W.L. Chang, M. Guo, and M. (Shan-Hui) Ho. Fast parallel molecular algorithms for DNA-based computation: factoring integers. *IEEE Trans Nanobiosci*, 4(2):149–163, 2005.
12. D. Eppstein. Subgraph Isomorphism in planar graphs and related problems. Tech. Report 94-25, Department of Information and Computer Science, University of California, Irvine, CA, 1994.
13. D. Faullhammer, A.R. Cukras, R.J. Lipton, and L.F. Landweber. Molecular computation: RNA solutions to chess problems. *Proc Natl Acad Sci USA*, 97:1385–1389, 2000.
14. R.P. Feynman. *In Minaturization*, D.H. Gilbert, editor, Reinhold, New York, 1961, pp. 282–296.
15. M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness*. W.H. Freeman, New York, 1979.
16. S.D. Gillmor, P.P. Rugheimer, and M.G. Lagally. Computing with DNA on surfaces. *Surf Sci*, 500:699–721, 2002.
17. F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add. *Science*, 273:220–223, 1996.
18. M. Guo, M. (Shan-Hui) Ho, and W.L. Chang. Fast parallel molecular solution to the dominating-set problem on massively parallel bio-computing. *Parallel Comput*, 30:1109–1125, 2004.
19. S.Y. Hsieh and M.-Y. Chen. A DNA-based solution to the graph isomorphism problem using Adleman-Lipton model with stickers. *Appl Math Comput*, 197(2):672–686, 2008.
20. S.Y. Hsieh, C.W. Huang, and H.H. Chou. A DNA-based graph encoding scheme with its applications to graph isomorphism problems. *Appl Math Comput*, 203(2):502–512, 2008.
21. R.J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.
22. Q. Liu, L. Wang, A.G. Frutos, A.E. Condon, R.M. Corn, and L.M. Smith. DNA computing on surfaces. *Nature*, 403:175–179, 2000.
23. Q. Ouyang, P.D. Kaplan, S. Liu, and A. Libchaber. DNA solution of the maximal clique problem. *Science*, 278:446–449, 1997.
24. G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms*, Springer-Verlag, Berlin, Germany, pp. 117–149.

25. M.J. Perez-Jimenez and F. Sancho-Caparrini. Solving Knapsack problems in a sticker based model, 2nd Annual Workshop on DNA Computing, DIMACS: series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, Providence, RI, 2001, pp.161–171.

26. S. Roweis, E. Winfree, R. Burgoyne, N.V. Chelyapov, M.F. Goodman, P.W.K. Rothemund, and L.M. Adleman. A sticker based model for DNA computation. *Proceedings of the Second Annual Worksop on DNA Computing*, *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 1996, pp. 1–29.

27. K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya. Molecular computation by DNA hairpin formation. *Science*, 288:1223–1226, 2000.

28. R.R. Sinden. *DNA Structure and Function*. Academic Press, New York, 1994.

29. S-Y. Shin, B-T. Zhang, and S-S. Jun. Solving travelling salesman problems using molecular programming. *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, Volume 2, 1999, pp. 994–1000.

# II

# ANALYSIS OF BIOLOGICAL SEQUENCES

# 10

# GRAPHS IN BIOINFORMATICS

Elsa Chacko and Shoba Ranganathan

## 10.1 GRAPH THEORY—ORIGIN

Graph theory emerged in 1736 when Euler addressed the problem of walking across the seven bridges of Königsberg without crossing any bridge twice [1]. Euler used the benefits of graph theory to conclude that it was impossible to walk through the city crossing each bridge only once. A century later, graphs were applied to recreational mathematical problems [2] such as the Knight's Tour and the Icosian Game [3]. Representing graphs in the form of dots and lines emerged out of 19th century chemistry, with the introduction of the term *graph* into both the chemical and mathematical literature by Sylvester [4], with a molecule represented by the connectivity between its constituent atoms. Since then, graphs have been applied successfully to diverse areas such as chemistry, operations research, computer science, electrical engineering, and drug design. More recently, graph theory has been used extensively to address biological problems. After a brief introduction to graph theory and the generic solution set commonly applied to several fields, we present select recent applications of significance in bioinformatics.

### 10.1.1 What is a Graph?

A *graph* is a set of nodes or vertices connected by a set of links, connections, or edges. This is represented mathematically as $G = (V, E)$, where $V$ represents the *vertices* and $E$ represents the *edges* [5]. Two vertices are said to be *adjacent* if there is an edge connecting them.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

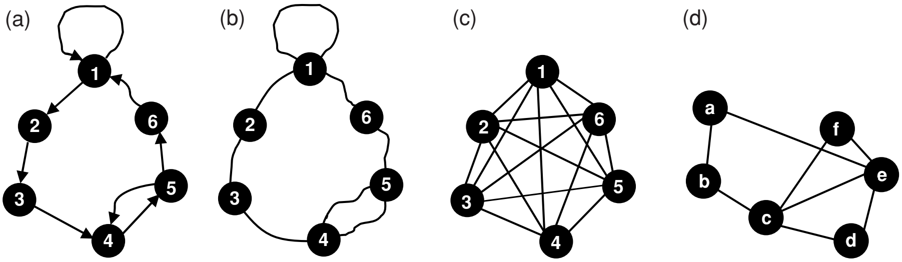


**Figure 10.1** Types of graphs. (a) *Directed graph*. All edges are directed. (b) *Undirected graph*. No edges are directed. (c) *Complete graph*. Every vertex is connected to every other vertex with exactly one edge. (d) Example of a generic graph. Here, a *walk* defines the path from one vertex to another (*e.g.*, *a-b-c-d-e*); a *trail* is a walk with no repeated edges, (*e.g.*, *a-b-c-f-e-c-d*), a *path* is a trail with no repeated vertices (*e.g.*, *a-b-c-e-d*), a *circuit* is a closed trail (*e.g.*, *a-b-c-f-e-c-d-e-a*), and a *cycle* is a closed path (*e.g.*, *a-b-c-d-e-a*).

### 10.1.2 Types of Graphs

Graphs either can be directed or undirected. In a *directed graph* (*digraph*), each element of $E$ (edges) is an ordered pair (a,b). An ordered pair implies that (a,b) $\neq$ (b,a) unless $a = b$, where $a$ and $b$ are elements of the set of vertices $V$, $a$ being the initial vertex and $b$ being the terminal vertex of an edge [5]. In other words, a directed graph is simply a graph whose edges have a specific direction. A *loop* is an edge that starts and ends at the same node. Figure 10.1a depicts a directed graph with $V = \{1, 2, 3, 4, 5, 6\}$. The edge set is $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1), (5, 4), (1, 1)\}$. Vertex 1 carries a loop, whereas vertices *4* and *5* have *multiple edges* between them. In an *undirected graph*, the edges do not have directionality. Each edge, $\{a, b\}$, is an element of the set $E$, such that $\{a, b\} = \{b, a\}$. Figure 1b shows an undirected graph that has the same set of vertices as the directed graph (Figure 1a). However, in the undirected graph, $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\},\{5, 6\}, \{6, 1\}, \{1, 1\}\}$. Although there are multiple edges between the vertices *4* and *5*, both are represented in the edge list as $\{4, 5\}$ because $\{4, 5\} = \{5, 4\}$, making them redundant. Because of this redundancy reduction, undirected graphs usually end up with fewer edges than directed graphs with the same number of nodes.

Graphs also either can be simple or complex. A graph in which there are no self loops and only one edge between two vertices is called a *simple graph*, whereas all others are called *multigraphs* [6]. A road map and a flowchart are examples of multigraphs. A graph in which every vertex is connected to every other vertex with exactly one edge is called a *complete graph* (Figure 1c).

***10.1.2.1 Walks, Trails, Paths, and Cycles.*** Translocating from one vertex to another vertex along an edge is called a *walk*. The number of edges making up the walk defines the *length* of the walk. If no edges occur more than once, then the walk becomes a *trail*. A *path* is a special representation of a trail with no vertex being repeated. A walk in which the initial vertex is the same as the final vertex

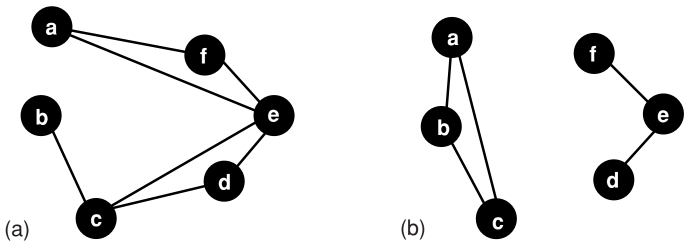


**Figure 10.2** Types of graphs. (a) *Connected graph.* (b) Two-component *disconnected graph.*

is called *closed*. A *cycle* is a closed path, whereas a *circuit* is a closed trail [7]. In Figure 10.1d, the traversal, *a-b-c-d-e* is one example for a walk through the graph. The walk *a-b-c-f-e-c-d* is a trail, as none of the edges are repeated. However, *a-b-c-e-d* is a path because none of the vertices have been repeated. The walk, *a-b-c-f-e-c-d-e-a* is a circuit, whereas *a-b-c-d-e-a* is a cycle, although both have the same initial and terminal nodes. In general, the terms path and trail refer to a walk through a given graph.

A graph in which there is a path from every vertex to every other vertex is a *connected graph*. Figure 10.2a represents a connected graph, despite the fact that vertex $a$ is not directly connected to vertex $b$. *a-f-e-d-c-b* is a path from vertex $a$ to vertex $b$. On the other hand, Figure 10.2b represents a disconnected graph, as there is no path from node $a$ to node $f$.

A graph with no cycles is *acyclic* (Figure 10.3a), whereas a digraph with no cycles is a *directed acyclic graph* (DAG; Figure 10.3b). On the other hand, a graph that has at least one cycle is called a *cyclic graph* (Figure 10.3c).

A *weighted graph* has values associated with its edges (edge-weighted) or vertices (vertex-weighted). If the graph is directed, it would then be called a *weighted directed graph* [5]. Undirected graphs also can have weights associated with them. Figure 10.4 shows a *weighted undirected graph* in which each edge is associated with a number, representing the weight of the edge.

If $v$ is the vertex of an undirected graph, then the degree of $v$ is the number of edges connected to it. If $v$ is a directed graph, then the out-degree of $v$ is the number of edges commencing at $v$, whereas the *in-degree* of $v$ is the number of edges terminating at $v$. A *regular graph* is a graph in which all vertices have equal degrees.

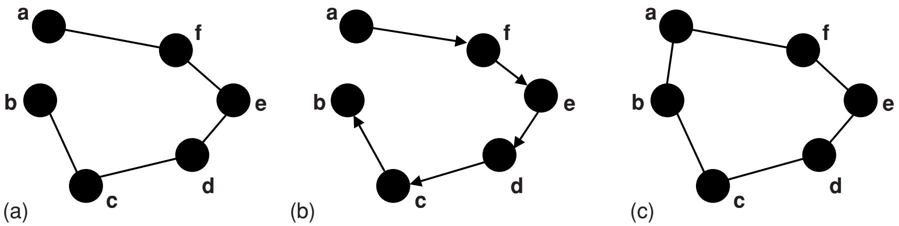


**Figure 10.3** Types of graphs. (a) *Acyclic graph.* (b) *Directed acyclic graph.* (c) *Cyclic graph.*

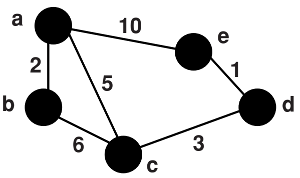


**Figure 10.4** *Weighted graph*. Every edge has a value associated with it.

A vertex in a directed graph is *balanced* if its in-degree equals its out-degree [7]. In Figure 10.1b, the degree of the vertex 4 is three. In Figure 10.1a the in-degree of the vertex 4 is two and the out-degree is one. Vertex 6 in Figure 10.1a is balanced because its in-degree is equal to its out-degree.

#### 10.1.2.2 Eulerian Graph.

*Eulerian graphs* are based on Euler's theorem, which stemmed from the concept that no walking tour of the seven bridges of Königsberg can be designed such that each bridge is traversed exactly once. Figure 10.5a shows the seven bridges of Königsberg and Figure 10.5b shows the graphical representation of these seven bridges. The four land masses are represented as nodes (*A, B, C, D*), and the bridges connecting the land masses are represented as edges. An *Eulerian path, trail*, or *walk* through a graph is a path whose edge list contains each edge of the graph exactly once [5]. If the graph is a circuit/cycle, then it is an *Eulerian circuit* or *Eulerian cycle/tour*. An *Eulerian graph* is a graph with an Eulerian path.

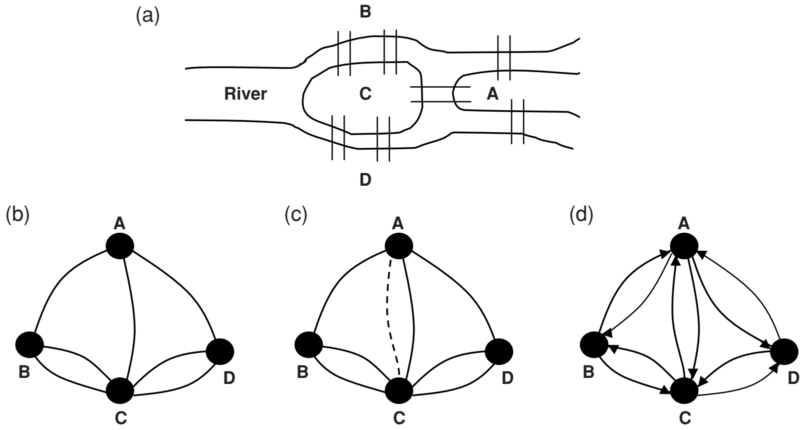


**Figure 10.5** Eulerian and Hamiltonian paths. (a) Schematic diagram of the seven bridges of Königsberg. (b) Graphical representation of the seven bridges of Königsberg. All vertices have odd degrees and therefore do not satisfy the condition for an Eulerian graph. This is, however, a Hamiltonian graph, and the path *A-B-C-D* is a Hamiltonian path. (c) Adding a second edge *A-C* makes the graph in Figure 10.5a Eulerian. Only vertices B and D have odd degrees. The path *B-C-B-A-C-A-D-C-D* is an Eulerian path. (d) A directed Eulerian graph. The in-degree of every vertex is equal to its out-degree. This is also a Hamiltonian graph with *D-A-B-C-D* being a Hamiltonian circuit.

An undirected graph is Eulerian if and only if it is connected and has either zero or two vertices with an odd degree. If no vertices have an odd degree, then the graph has an Eulerian circuit.

It can be observed from Figure 10.5b that the seven bridges of Königsberg have odd degrees on all vertices (three on *A, B*, and *D* and five on *C*), and therefore, this graph is not Eulerian. However, adding one more edge between the vertices *A* and *C* makes the degrees of *A* and *C* even (Figure 10.5c) [5]. This new graph now has only two vertices with odd degrees (*B* and *D*), and therefore, it is possible to find an Eulerian path by starting at one of the two odd-degree nodes, such as *B-C-B-A-C-A-D-C-D*. In a directed graph, if the in-degree of every vertex is equal to its out-degree, then it is sure to have an Eulerian circuit. In Figure 10.5d, the in-degree is the same as the out-degree for every node, and *A-B-C-D-C-B-A-C-A-D-A* is an Eulerian circuit.

**10.1.2.3 Hamiltonian Graphs.** The next interesting problem in graphs is to find a path in which each node is visited only once. Hamilton defined graphs in which each node is visited exactly once now are known as *Hamiltonian graphs*. The vertex list of a Hamiltonian path through a graph contains each vertex exactly once. A circuit fulfilling this criterion is a Hamiltonian circuit. A *Hamiltonian graph* is one that has a *Hamiltonian path* [8]. The path *A-B-C-D* in Figure 10.5b is a Hamiltonian path, whereas the path *D-A-B-C-D* in Figure 10.5d is a Hamiltonian circuit.

**10.1.2.4 Subgraphs and Cliques.** A *subgraph* is a smaller portion of the original graph. Every vertex and edge of the sub-graph is a member of the vertex and edge list, respectively, of the main graph. For a graph $G = (V,E)$, a subgraph is defined as $S = (V', E')$, where $V'$ is a subset of $V$ and $E'$ is a subset of $E$. Figure 10.6a is a subgraph of the complete graph in Figure 10.1c. We can see that in the subgraph, some edges and vertices are missing, thereby making it a smaller version of the original graph. In an undirected graph, a *clique* is a subgraph

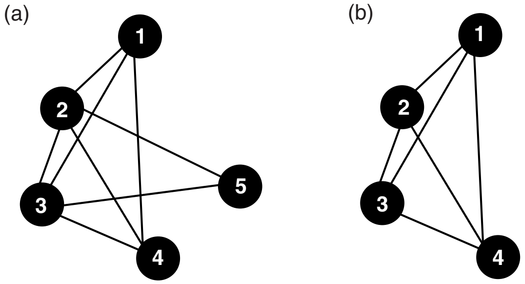


**Figure 10.6** Subgraphs and cliques. (a) A s*ubgraph* of the graph in Figure 10.1c. It is evident that the edge list of the graph in (a) is a subset of the edge list of the graph in Figure 10.1c. However, the vertex list is the same for both graphs. (b) A *complete subgraph* of (a) in which every vertex is connected to every other vertex. This subgraph is a *clique* of size 4. Vertex *5* in (a) is absent.

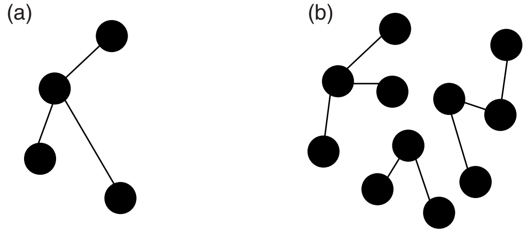


**Figure 10.7** Trees and forests. (a) A *tree*. The graph is connected but undirected. (b) A *forest* of trees. Each separate component of the forest is connected and is a tree.

*C* = (*V'*, *E'*), comprising pairwise edges. In other words, the subgraph *C* is a complete graph. The *size* of a clique is measured by the number of vertices it contains [9]. Figure 10.6b is a complete subgraph of the graph in Figure 10.6a and is thus a clique of size four.

***10.1.2.5 Trees, Forests and Spanning Trees.*** A connected graph that has no self-loops or cycles is called a *tree*. A *forest* is an undirected graph whose components are all connected. Figure 10.7a is a tree and Figure 10.7b is a forest. It can be seen that a forest is a disconnected graph whose individual components are connected. In a tree, if there is a unique path from a particular vertex to every other vertex, then this special node is considered distinguished and is referred to as a *root*.

A tree with a distinguished vertex is a *rooted tree*. The *level* of a vertex in a rooted tree is the number of edges that separates the vertex from the root. In Figure 10.8, *A, C, D, E* and *F* are *child* nodes of the root vertex *B*. *D* is a *parent* of both *G* and *H*, whereas *B* and *D* are *ancestors* of *G*. *G* and *H* are *descendents* of *D*. Any vertex that does not have descendents is called a *leaf* (*A, C, E, F, G*, and *H* in Figure 10.8). The vertices *D, G*, and *H* form a *subtree* of *B*. Vertices *E, F, A*, and *C* are also subtrees with only one node each. A *binary tree* has only two subtrees for any vertex at each level.

A *spanning tree*, *T* = (*V, E'*) of the graph, *G* = (*V, E*) is a tree that connects all vertices of *G* such that *E'* is a minimum subset of *E* required to connect every vertex

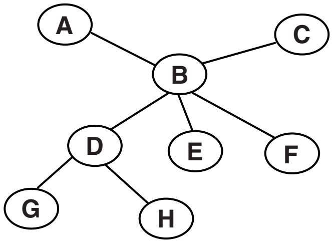


**Figure 10.8** A *rooted tree*. Vertex *B* is the root node.

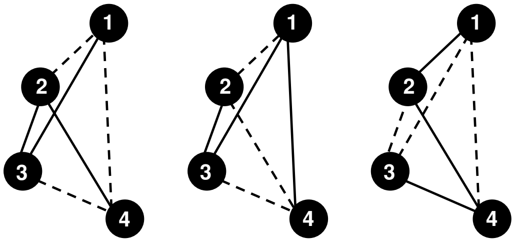


**Figure 10.9** Three possible trees with three edges are shown, with *spanning trees* shown in dotted lines. It is possible to find many spanning trees for the same graph.

in V [5]. Figure 10.9 shows three different possibilities of spanning trees (dotted lines) for the graph in Figure 10.6b. As the number of vertices and edges increases, it is almost impossible to find all possible spanning trees for a given graph, without the aid of a computer.

***10.1.2.6 Networks and Flows.*** A *network* is a weighted digraph that contains two distinguished vertices called the *source* and the *sink*. The maximum rate of *flow* through any edge is called its *capacity*, and it is this information that the weight function of the network contains. The flow through a network can be defined as a function $f$ such that the amount of information passing through any edge is nonnegative and is not larger than its *capacity* [5]. For each vertex other than the source or the sink, the amount of information going into the vertex is equal to the amount of information leaving the vertex. Figure 10.10a shows a network in which $s$ is the source and $t$ is the sink. Figure 10.10b represents one flow through the network (dotted lines) from source to sink. The amount of flow along each edge is represented as the first number and the capacity of the edge is the second number. The flow along an edge should always be less than or equal to the capacity of the edge. There can be many other flows through a given network from source to sink.

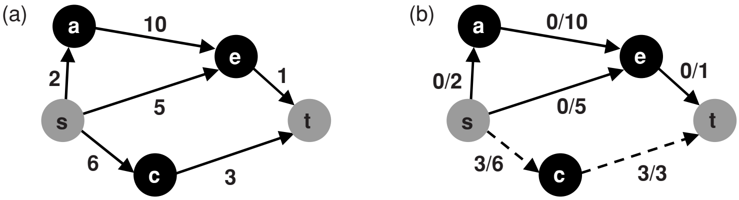


**Figure 10.10** *Network, flow*, and *capacity*. (a) Each *network* has a source *s* and a sink *t*. Every edge has a weight associated with it that is called the *capacity* of the edge. (b) *s-c-t* is one of the many flows through the network starting at the source and ending at the sink. The flow from the source to the sink is 3.

### 10.1.3 Well-Known Graph Problems and Algorithms

Graphs are a convenient way of representing the relationship between entities, and therefore, there has been a lot of innovation in finding new algorithms to represent different problems in everyday life. A few of these algorithms have been discussed in this chapter.

***10.1.3.1 The Traveling Salesman Problem (TSP).*** The notion of the traveling salesman problem (TSP) originated from the situation of a salesman who wants to minimize the distance that he travels to visit all his customers. In other words, a *traveling salesman problem* is a Hamiltonian path in which the sum of the weights of the edges is a minimum. It is possible to find numerous Hamiltonian paths through a given graph; however, to identify the *minimum cost* or *optimum solution*, an exhaustive check to find the cost of every possible path is required. As the number of vertices in a graph increases, this method becomes nearly impossible to determine manually. Harnessing computational power to find the optimal solution has resulted in the optimal TSP solution to be calculated within a few hours for 1000 cities and in a few seconds for 100 cities [10]. Many heuristic algorithms have been developed to find a path or circuit that is closest to the optimal one. One such algorithm is the *closest neighbor algorithm*. This algorithm starts at any random vertex *a* and visits the closest neighbor that has not been visited previously. At each vertex, the next vertex to be visited is such that the distance from the current vertex to the next unvisited vertex is the smallest available one. In the completely connected and weighted graph in Figure 10.11, the TSP problem [5] has been solved using the closest neighbor algorithm. Vertex *a* is selected randomly as the first node. The possible traversals from *a* include *a-c*, with weight five, *a-b*, with weight six, and *a-d*, with weight eight. As five is the smallest of the three, *a-c* is selected as the first edge. The next choice is either *c-b* (six) or *c-d* (seven). *c-b* is thus the selected route. The two subsequent visits from *b* are either to *a* or *d*. As vertex *a* is already visited, the only choice is to travel to *d* and then to a to complete the circuit. The final path covering all vertices has been shown in dotted lines (*a-c-b-d-a*) with a cost of 29.

However, if checked carefully, we can find another circuit (*a-b-c-d-a*) in this graph that has a smaller cost of 27. It is not always possible to find an optimal solution with

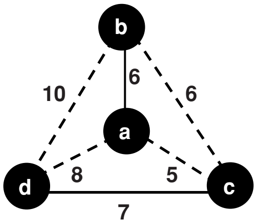


**Figure 10.11** The TSP has been solved for the given graph using the closest neighbor algorithm. Vertex *a* is selected randomly. The nearest node to *a* is node *c*. The nearest node to *c* is *b* and the nearest node to *b* that is not visited is *d*. Finally, the circuit is completed by travelling back to node *a* (starting node). The edges traversed from each node is shown in dotted lines.

the closest neighbor algorithm. Therefore, we have to try and find a closest neighbor circuit in a graph such that the ratio of the cost of the closest neighbor path ($C_{\text{cn}}$) to that of the optimal path ($C_{\text{opt}}$) is as close to 1 as possible. If $\frac{C_{\text{cn}}}{C_{\text{opt}}}$ is $\sim 1$, then that algorithm is considered good [5].

#### *10.1.3.2 The Minimal Spanning Tree Problem (MST).*

For a weighted, connected and undirected graph, $G = (V, E, w)$, where $V$ is the vertex list, $E$ is the edge list and $w$ is the list of weights associated with the edges, a *Minimum spanning tree* (MST) is $T = (V, E')$, where $E'$ is a subset of $E$, such that there are no cycles in $T$, but every vertex is connected. The sum of the weights of each edge in the spanning tree ($E'$) should be as small as possible [8]. In other words, a spanning tree of a connected graph $G$ also can be defined as a minimal set of edges of $G$ that connects all vertices. Two commonly used approaches to find minimal spanning trees are Kruskal's algorithm and Prim's algorithm. Both these algorithms generate the same minimal spanning tree for a given graph. In total, we will have to find n–1 edges to find a minimal spanning tree for a graph with $n$ vertices [6].

##### *10.1.3.2.1 Kruskal's Algorithm.*

In Kruskal's algorithm [11], a given graph first is broken down into a forest that consists of $n$ components, where $n$ is the number of the vertices of the given graph. Each component in the forest consists of only one node and nothing else. Next, an edge with the smallest weight is selected such that the selected edge does not create a loop, and it does not connect vertices that belong to the same tree. This edge is added to the new list of edges for the spanning tree. This procedure is continued until all vertices are visited, resulting in only one tree. At every step of the algorithm, two different trees of the forest are connected to make a bigger tree. Therefore, from smaller trees in our forest, we end up with a tree that is the minimal spanning tree [12].

By applying the Kruskal's algorithm to the graph in Figure 10.12a we can obtain the minimal spanning tree in Figure 10.12b (dotted lines). Of the two edges *A-E* and *B-E*, with weights of only two (the smallest weight in the graph), let *A-E* be selected. The next shortest arc that does not form a cycle is *B-E*, and is added to the tree. *E-C* with a weight of three then is added to the new edge list followed by *E-G*. The subsequent smallest weight is five between nodes *B* and *A*. This edge is not selected as it forms a cycle, *B-E-A-B*. The edge, *B-C*, has a weight of six but is ignored, as *B* and *C* are already part of the same tree. The next edge selected is *B-D*, with a weight of seven. The only vertex left to connect is *F*, which is connected to either *C* with a cost of eight or *G* with a cost of 11. We select *F-C*, which is the smaller of these two cost values. Therefore, the final spanning tree has six edges, connects all vertices, and has a total cost of 26. It is also important to note that the total number of edges in a spanning tree is one less than the number of nodes in the given graph.

##### *10.1.3.2.2 Prim's Algorithm.*

Jarník (1930), Prim (1957), and Dijkstra (1959) are believed to have developed this algorithm independently, although it is attributed to Prim in current literature [12]. As discussed earlier, for a given connected weighted

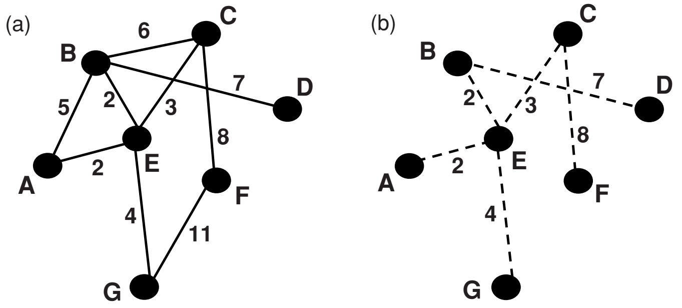


**Figure 10.12** Applying Kruskal's algorithm. (a) A *weighted connected undirected graph*. (b) The minimal spaning tree for the graph obtained by applying the Kruskal's algorithm. Edges *A-E* (2), *B-E* (2), *E-C* (3), *E-G* (4), *B-D* (7), and *C-F* (8) constitute the minimum spanning tree, of weight 26. The same minimal spanning tree is obained by applying Prim's algorithm with any vertex chosen as the starting point.

graph $G = (V, E, w)$, a minimal spanning tree is defined as $T$ = (V, E'). Prim's algorithm finds the minimal spanning tree $T$ as follows:

1. Chose an arbitrary vertex, say $x$, from $V$ and add this to $V_{new}$. So $V_{new} = \{x\}$. No matter which vertex is selected, the solution will be the same.
2. Select an edge ($x$, $y$) so that $x$ is a member of $V_{new}$ and $y$ is an unvisited vertex that has the least weight. If there are multiple edges with the same weight, then choose arbitrarily but consistently.
3. Add this newly visited vertex to $V_{new}$ and the edge ($x$, $y$) to $E_{new}$.
4. Repeat steps 2 and 3 until $V_{new} = V$.

In short, the algorithm works by choosing a random starting node and building a tree by selecting at every stage the shortest available edge starting from any of the already visited vertices that can extend the tree to an additional node [6].

Figure 10.12b represents the minimal spanning tree obtained by applying Prim's algorithm to the graph in Figure 10.12a. We have selected $A$ as the first node in $V_{new}$. There are two possible edges that can be selected from $A$, namely $A$-$E$ (2) and $A$-$B$ (5), where the values in the brackets represent their respective costs. $A$-$E$ has the minimum cost, and therefore, this edge is chosen and is added to the edge list, and $E$ is added to $V_{new}$. The next edge chosen can be any connected to $A$ or $E$. $E$-$B$ (2) is subsequently selected. Thereafter, $E$-$C$ (3) is the least weighted edge among the different edges possible from the already visited vertices $A$, $E$, and $B$. Next, the edge $E$-$G$ (4) is chosen. Although the next smallest edge available is $A$-$B$ (5), this edge is ignored as both $A$ and $B$ are already in $V_{new}$. The same argument holds for edge $B$-$C$. So, the next edge selected is $B$-$D$ (7). The only node left to be connected is $F$, which is linked to $C$ (8) or to $G$ (11). We chose $C$-$F$, as this has a smaller weight than $F$-$G$. The resulting MST is identical to the one obtained from Kruskal's approach.

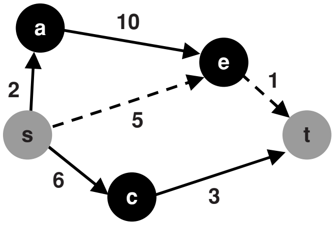


**Figure 10.13** Shortest path through the graph from the source to the sink is *s-e-t* that has a weight of six.

***10.1.3.3 The Shortest Path Problem.*** The *shortest path problem* involves finding a path between two given vertices such that the sum of the weights of its edges is a minimum [13]. One of the most important approaches to solve this problem is Dijkstra's algorithm. An everyday example is finding the quickest way to get from one location to another on a road map. In this case, the vertices represent locations, and the edges represent segments of road and are weighted by the time needed to travel each segment. Figure 10.13 shows the shortest path (dotted lines) through a connected and weighted graph from the source, *s* to the sink, *t*.

*10.1.3.3.1 Dijkstra's Algorithm.* Dijkstra's algorithm, developed in 1959 [14], can be applied to solve the single-source shortest path problem for a graph with nonnegative edge costs, resulting in the shortest path tree. For a given source vertex (*s*) in the graph, the algorithm finds the path with the least cost between *s* and every other vertex.

The node that we start with is called an *initial node*. Dijkstra's algorithm first assigns arbitrary initial distance values to each node and then attempts to improve these values iteratively [6].

1. Select an initial node and assign a distance value of zero to it. Assign a distance of infinity to all other nodes.
2. Mark the initial node as current (white) and all other nodes as unvisited.
3. For the current node, calculate the distance to all its unvisited nodes from the initial node. If this distance is less than the previously recorded distance, then overwrite the distance.
4. Mark the current node as visited (grey) after considering all its neighbors. A visited node will not be checked again; its distance recorded now is final and minimal.
5. Set the unvisited node with the smallest distance from the current node as the next current node and repeat steps 3–5.

Figure 10.14 shows an example of Dijkstra's algorithm being carried out. *a* is the initial node, and *d* is the final node, in a graph with five vertices (Figure 10.14a). First, node *a* is assigned a distance value of 0 and marked as the current node, whereas all

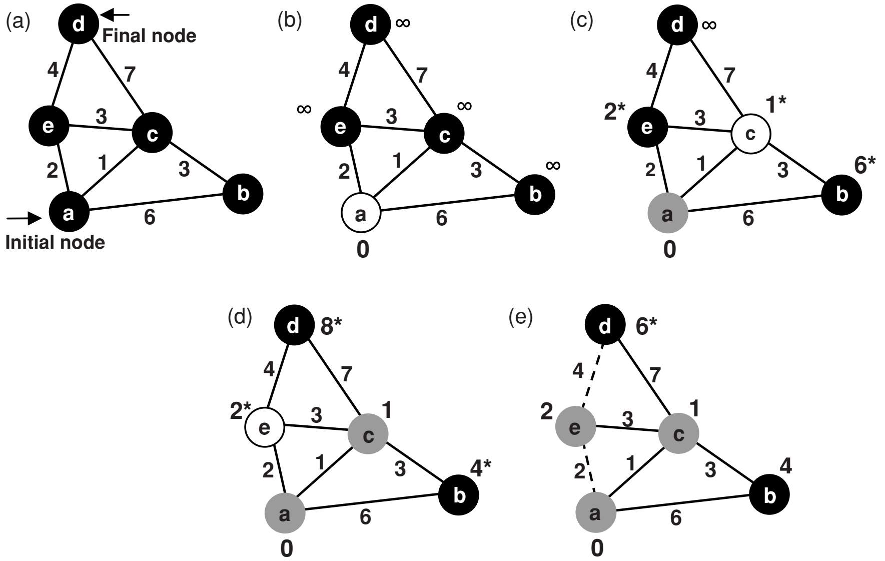


**Figure 10.14** Using Dijkstra's algorithm to find the shortest path. (a) The initial node is node *a*, and the final node is node *d*. (b) The initial node (white) is assigned a distance value of 0, and all other nodes are assigned a distance value of $\infty$. (c) Node *a* is marked as visited (grey) after calculating the distance to all unvisited neighboring nodes *e*, *c*, and *b* (marked with *). *c* is set as the next current node (white). (d) Node *c* is marked as visited (grey) after calculating the distance to all unvisited neighboring nodes *e*, *d*, and *b*. Note that the distance to node *b* has been overwritten as the new distance value is less than six (the previous value on *b*). *e* is set as the next current node (white), as it is the unvisited node with the smallest distance from node *c*. (e) Node *e* is marked as visited after calculating the distance to the only unvisited neighboring node *d*. The shortest path to *d* from *a* (initial node) therefore, is *a*-*e*-*d* with a distance value of six.

other nodes are assigned distance values of infinity (Figure 10.14b). Node *a* has three neighbors: *b*, *c*, and *e*. The distances to the three nodes *b*, *c*, and *e* are calculated as the weights of the interconnecting edge, and the infinity value is replaced by the current values of six, one, and two, respectively (Figure 10.14c). Node *a* is then marked as visited, and node *c*, which is closest to node *a*, is marked as the new current node, with four neighboring nodes, *a*, *e*, *d*, and *b* (Figure 10.14d). Because *a* is already marked as visited, this node is ignored and the distance values to the remaining three nodes are calculated from *a* by summing up the distance *a*-*c* and the distance between *c* and each of its other neighbors.

Applying this formula, the distance from *a* to *e* through *c* is *a*-*c* (1) + *c*-e (3) = 4. The distance value of *e* is not overwritten as the new distance value of *e* is $4 > 2$, the existing value on *e*. But the distance from *a* to *b* through *c* is *a*-*c* (1) + *c*-*b* (3) = 4, less than the previously recorded value of six. So we replace six with the newly obtained distance value for *b*. Similarly, the distance from *a* to *d* through *c* is eight. The nearest node to *c* is *e*, based on its weight, making it the next current

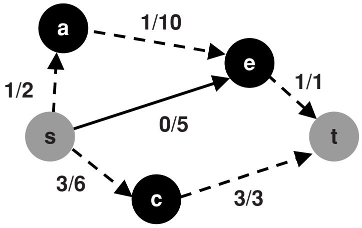


**Figure 10.15** Maximal flow network for the graph in Figure 10.13. The first value shown on each edge is the actual flow through the network, and the second value is the edge's capacity. The flow from the source *s* to nodes *a* and *c* is 1 and 3, respectively, resulting in the total flow leaving the source being 4. The total flow entering the sink *t* is also therefore 4. The edges participating in the maximal flow are shown in dotted lines.

node. The new distance value for *d* (six) is smaller than the previous value (eight) and therefore is replaced with the new value. Thus, the shortest path from *a* to *d* has a weight of six (dotted lines) in Figure 10.14e.

***10.1.3.4 The Maximal Flow Problem.*** The *maximal flow problem* involves maximizing the amount of information that passes through a network [6]. This is the same as finding the flow with the largest possible value in any given network. Figure 10.15 shows the maximal flow network for the graph in Figure 10.15. There are two values represented on the edge in a network; the first value being the actual flow *f* through the edge, and the second value being the capacity *c*. Several algorithms are available to calculate the maximal flow in a directed graph in which each edge has a defined capacity. One such algorithm is the *Ford–Fulkerson algorithm.*

*10.1.3.4.1 Ford–Fulkerson Algorithm.* The *Ford–Fulkerson algorithm*, published in 1956 [15], computes the maximum flow in a network. To describe the Ford–Fulkerson algorithm, we need to define *residual networks* and *augmentation paths* [16]. If the flow *f* along an edge *a-b* in a given network is less than the capacity *c* of the edge, then there is a forward edge *a-b* with a capacity equal to the difference of the capacity and the flow (= *c-f*), known as the *residual capacity*. If the flow is positive, then there is a backward edge, *b-a* with a capacity equal to the flow (= *f*) on *a-b*. An *augmenting path* links the source to the sink in the residual network and helps to increase the flow in a network. In an augmenting path, the edges can point the wrong way when compared with the real network. The minimum flow possible from source to sink in the given network is determined first. Then a *residual network* of this graph is found to check whether there is an augmentation path in this residual network. If there is a path, then more flow is directed along this path. The process is continued untill no augmentation path is available in the residual network from source to sink.

In Figure 10.16a, there are a few different paths from the source *s*, to the sink *t*. Let us select the path *s-a-e-t*. The capacity of the edges *s-a*, *a-e*, and *e-t* are two, ten, and one, respectively. We select the smallest capacity (one) along this path and

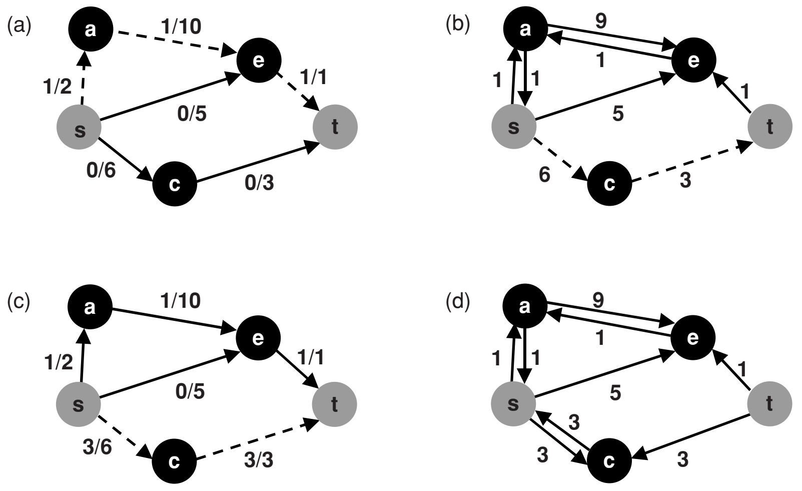


**Figure 10.16** Using the Ford–Fulkerson algorithm to find the *maximal flow* for a network from source *s* to sink *t*. (a) A flow of 1 is directed along *s-a-e-t*. (b) The *residual network* Figure 10.16a. As *s-a* in the orginal network has a flow of 1, the forward edge *s-a* in the residual network has a capacity of 1 (difference of capacity(2) and flow (1)). The flow is positive from *s* to *a*. So the backward edge *a-s* is equal to the actual flow (1) through the edge. Likewise, the forward edge *a-e* has a capacity of 10 − 1 = 9 the backward edge *e-a* has a capacity of 1. The forward edge *e-t* has a capacity of 0 as flow-capacity on this edge is zero. Edges *s-c*, *s-e*, and *c-t* also have backward edges with capacity zero as flow = 0 on these edges. There is an *augmentation path* (*s-c-t*) from *s* to *t* in this residual path. (c) A flow of 3 (minimum capacity of the edges involved) is directed along the augmentation path, thereby increasing the flow from 1 to 4. (d) Residual network of Figure 10.16c. There is no available augmentation path in this residual network from *s* to *t*. The maximal flow through the network is 4.

direct this amount of flow through that path as depicted in Figure 10.16a. Therefore, in the residual network (Figure 10.16b), edge *s-a* has a forward and backward flow of capacity one. The forward edge *s-a* has a capacity of one (difference of capacity (two) and flow (one)). As the flow is positive from *s* to *a* in Figure 10.16a, the backward edge *a-s* is equal to the actual flow (1) through the edge. Likewise, the forward edge *a-e* has a capacity of 10 − 1 = 9 the backward edge *e-a* has a capacity of one. The forward edge *e-t* has a capacity of zero, as flow is equal to capacity on this edge. Edges *s-c*, *s-e* and *c-t* have backward edges with capacity zero, as flow equals to zero on these edges in Figure 10.16a. We can see clearly from Figure 10.16b that there is an augmentation path (*s-c-t*) from *s* to *t* in this residual network. A flow of three (minimum capacity of the edges involved) is directed along the augmentation path, thereby increasing the flow from one to four (Figure 10.16c). There is clearly no augmentation path present in the new residual network (Figure 10.16d) from *s* to *t*. Therefore, the maximal flow through the network is four.

The veracity of this algorithm can be validated by using the *max-flow min-cut theorem* [17]. The theorem states that the *maximum flow* in a network is equal to the *minimum cut* of the network. A cut in a flow network is the set of edges such that if they are removed, then there is no path from the source to the sink. The capacity of a cut is equal to the sum of the capacities of the every edge in the cut. In Figure 10.15, *s-a*, *s-e*, and *c-t* together represent a cut with capacity 10. Similarly, *e-t* and *c-t* can be another cut with capacity four. By exhaustively determining every possible cut, we can see that the smallest possible cut for the graph in Figure 10.15 has a capacity of four. According to the max-flow min-cut theorem, the maximum flow through the network should be 4 for the graph. This is exactly the same value for maximum flow that we found using the Ford–Fulkerson algorithm in Figure 10.16.

## 10.2 GRAPHS AND THE BIOLOGICAL WORLD

The data explosion in biology in recent years [18] can be attributed to the sequencing and annotation of several genomes, coupled with the advances in high-throughput experimental screening techniques. We now have data on genomes, transcriptomes, proteomes, and interactomes along with genes, proteins, transcription factors, pathways, and regulatory networks. Bioinformatics, the application of computational techniques to analyze the information associated with biomolecules on a large scale, now has established itself firmly as a discipline in molecular biology and encompasses a wide range of subject areas from structural biology and genomics to gene expression studies [19]. The language of graph theory offers a mathematical abstraction for the description of various relationships involved in molecular biology [20]. We present here recent bioinformatics applications of graph theory that proved their utility in analyzing large datasets.

### 10.2.1 Alternative Splicing and Graphs

*Alternative Splicing* (AS) is a fundamental mechanism that leads to complexity in higher eukaryotes. The introns in the pre-mRNA are removed in a process called *splicing*, and the exons are coupled in varying combinations. This can change the composition of the primary transcript. A single gene therefore can generate several unique transcripts by combining exons and introns in different combinations because of the phenomenon of AS. It is critical to conduct an indepth study on AS because the disruption of AS is associated with many diseases such as cardiovascular, cancer, and neurodegenerative disorders [21]. Analyses also have shown that up to 15% of all point mutations causing human genetic diseases result in an mRNA splicing defect [22], providing a link between AS events and inherited genetic diseases.

Splicing graphs facilitate the systematic study of alternatively spliced genes of higher eukaryotes by generating graphs for the compact visual representation of transcript diversity from a single gene. Lee *et al.* [23] have developed a database of *Drosophila melanogaster* genes (DEDB), with alternative splicing information of all transcripts developing from each gene organized as a splicing graph. Their study used DAGs; (Figure 10.3b) to represent the transcripts, which were clustered based on overlapping genomic positions. Splicing graphs were constructed using these clusters of transcripts. In each cluster, exons and introns that had identical start and end positions were merged into nodes and connections, respectively to form the complete splicing graph. Lee *et al.* [23] suggested that by condensing all splice variants into a single graph, where each splice variant is a path through the graph, users quickly can establish the types and effects of various alternative splicing events present in the gene. The splicing graphs provided an efficient platform to break down the graphs further and identify the various alternative splicing events within the same genes. This group has discussed eight alternative splicing events in which the exons in a gene either are included or excluded from the transcripts in varying combinations leading to different patterns within the transcripts and also suggested that by using this method to represent transcripts, users can pick up bifurcations that denote alternative splicing events far quicker than for the traditional approach of presenting separate transcripts, wherein the user has to deduct AS events by visual analysis.

The *Alternative Splicing Graph Server* (ASGS) [21] is another approach to depict transcripts from a single gene as a splicing graph. Unlike the approach of Lee *et al.* [23] in which the first transcript was used as a reference to define AS events for all other transcripts, ASGS attempts to identify the path most traversed in the splicing graph. For a given set of transcripts, ASGS identifiesdistinct exons as the most commonly occurring exons at a genomic location so that all other exons are classified as variable exons. AS events therefore are defined with respect to these distinct exons. Figure 10.17 shows the use of directed acyclic graphs to represent exons and introns used by ASGS.

### 10.2.2 Evolutionary Tree Construction

The construction of an evolutionary tree is a very challenging problem in computational biology and has been studied extensively by many researchers. Korostensky and Gonnet [10] have applied the traveling salesman problem algorithm to reconstruct a correct evolutionary tree by defining a phylogenetic tree $T = (V, E)$ as a binary connected acyclic graph, where the vertices are denoted as $V$ and the edges as $E$. For a given set of protein sequences, their method constructs a tree with a minimum score, representing evolution measured in point accepted mutation (PAM)

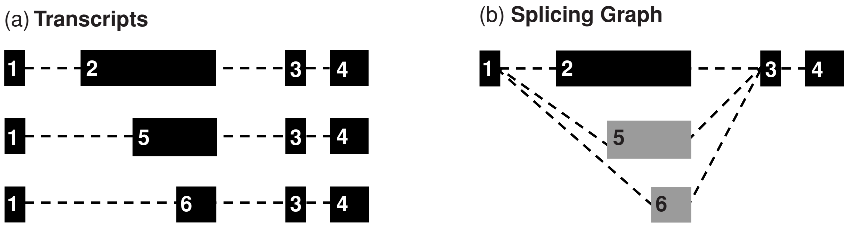


**Figure 10.17** An alternative splicing graph generated using ASGS. (a) Transcript structure of a single gene. The exons (black) are nodes and the introns (dotted lines) are edges. (b) The splicing graph representation of the gene. The distinct exons are shown in black and the variant exons are in grey.

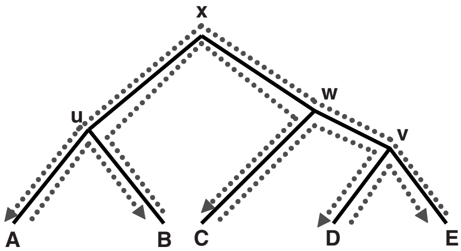


**Figure 10.18** Traversal of evolutionary tree in a circular order. The dotted lines represent the traversal. Evey edge is traversd exactly twice.

distances [10]. Their tree constructing algorithm, *CircTree* takes as input the PAM distances of the pairwise alignments as well as the circular order of the optimal tree calculated with a TSP algorithm. The *circular order* $C(T)$ for a set of sequences $S = \{s_1, \ldots, s_n\}$ is defined as any tour through a tree $T(S)$ in which each edge is traversed exactly twice, and each leaf is visited once. Figure 10.18 shows the traversal of a tree in a circular order along the dotted lines from *A-B-C-D-E-A*. It traverses all edges exactly twice and, thus, weighs all edges of the evolutionary tree equally.

Korostensky and Gonnet [10] were able to reconstruct a correct evolutionary tree if the error for each distance measurement was smaller than $\frac{x}{2}$, where $x$ is the shortest edge in the tree. For datasets with large errors, a dynamic programming approach was used to reconstruct the tree. The group carried out simulations with real data, and their studies showed that the algorithm produced good results compared with other established algorithms developed by Fitch and coworkers [24] and the *ProbModel* approach [25], especially for smaller trees.

### 10.2.3 Tracking the Temporal Variation of Biological Systems

A common approach to studying the behavior of naturally dynamic biological systems and phenomena is to sample individuals, tissues, or other relevant units at intervals throughout the chronological progression of the system under study [26], known as a time series. The temporal progress of biological systems can be measured by using well-characterized time series as benchmarks. Temporal variation data also can be used as diagnostic tools for assessing and treating diseases [26]. Magwene *et al.* used modifications of a minimal spanning tree calculated from a weighted undirected graph to estimate temporal orderings of biological samples from unordered sets of sample elements [26]. This technique was applied to an artificial dataset as well as to several gene expression datasets derived from DNA microarray experiments.

First a graph was constructed such that the vertices represent the sample observations, and the weights on edges are pairwise dissimilarities. Then, a minimum spanning tree $G_{\text{mst}} = \{V, E_{\text{mst}}\}$ of the complete weighted graph $G$ was found. This was taken to be the best estimate of the ordering if the minimum spanning tree obtained was a path and it had no branches.

When $G_{\mathrm{mst}}$ is not a path, the *diameter path noise ratio*, *branch distribution*, and *sampling intensity* were calculated. A noisy sample has numerous edges dangling from the diameter path, leading to these edges and the nodes they connect, forming branches. The diameter path noise ratio was calculated as the ratio of the number of points on branches to the total number of points. If the distribution of points on branches (known as the *branch distribution*) was uniform, then the branch points were truly noisy. If they were not uniform, then relatively long branches may represent signal rather than noise. The *sampling intensity* was calculated using the ratio of the average segment length to the total length of the diameter path. When the sampling seemed to be relatively intense for a relatively uniform diameter path branch distribution, the diameter path gave an estimate of the ordering.

A data structure called the *PQ-tree* was used to summarize all uncertainties of path variations if the diameter path sampling intensity ratio was large and the diameter path branch distribution seemed to be nonuniform with a few long branches coming off the diameter path. A secondary condition of the shortest path ordering was applied to the variations of the paths. Each of the $X$shortest paths that were consistent with the *PQ-tree* were reported, where $X$is a user-defined value.

This algorithm worked for both artificial and experimental datasets from bacterial and yeast gene expression studies. As the natural geometric characterization provided by minimum spanning trees are free of *a priori* distributional assumptions, this approach can be applied to any dataset using a variety of dissimilarity measures.

### 10.2.4 Identifying Protein Domains by Clustering Sequence Alignments

Protein domains are regarded as compact, independent units [27] that can fold separately [28] and perform a specific biological function in the living cell. Two proteins with similar domains exhibit similar functions, so that the identification of the functional domains of a protein serves as a means of protein annotation and classification [29].

The currently available tools for searching the enormous biological sequence databases result in inordinately long result files from which it is often difficult to glean biologically relevant functional information [30]. Weak but biologically significant homologies can be missed by not being able to identify many interesting relationships buried in numerous alignments in the output files. To parse such database similarity search output files, Guan and Du [30] algorithmically have classified matching sequences into clusters, where each of these clusters represents similarity to a different region of the query sequence, thereby extracting domain information. They construct a graph with sequences representing vertices and the distance between two sequences as edges. A small distance value is assigned to two sequences matching the same region of the query sequence and a greater distance to sequences matching different regions of the query sequence. For a query sequence of length $n$, the vector $C = c_1c_2 \ldots c_n$ represents the similarity of a database sequence to the query sequence in a search output, where $c_i = 1$ if the sequences are similar and $c_i = 0$ if they are dissimilar. The distance between two sequences is defined as the distance between their vector representations ($C^i$ and $C^j$), with small values if $C$ values are close to 1 and *vice versa*.

Prim's algorithm was used to find the MST in this connected graph. In a minimum spanning tree, if an edge is removed, separating the MST into two subtrees, then the two subtrees are still MSTs, with respect to the complete subgraphs formed by the nodes contained in the two subtrees [30]. First, the algorithm finds the edge with the longest distance in the tree and separates the tree by removing the edge if the two resulting subtrees differ by a constant $D$,

$$D < \frac{d(C^i, C^j)}{n}$$

where $d(C^i, C^j)$ is the difference between the vector representation of two sequences, $n$ is the length of the query sequence, and the number of nodes in each subtree is $\geq M$, where $M$ is a user-defined parameter. The use of $M$ helps to introduce some control over the cluster size (minimum cluster size) and limits the number of nodes that the clusters should have. The algorithm repeats this step until no more subtrees can be separated. Based on the constants $D$ and $M$, the algorithm classifies the sequences into several clusters. Guan and Du [30] observed that coarse clustering (high values of $D$ and $M$) reveal the main domains, whereas fine clustering (lower $D$ and $M$) reveals weak and distantly related domains.

### 10.2.5 Clustering Gene Expression Data

Functional relationships of genes in a biological process can be studied using gene expression data clustering. The problem of clustering genes with associated expression patterns over some time series and under different conditions requires efficient methods to interpret the observed expression data. Xu *et al.* [31] have suggested new methods to represent a set of multidimensional gene expression data as a minimal spanning tree using Kruskal's algorithm. They suggested that the simple structure of a tree facilitates the efficient implementation of rigorous clustering algorithms. Also, an MST-based clustering does not depend on the detailed geometric shape of a cluster and can overcome many problems faced by classical clustering algorithms. These clustering algorithms have been implemented in the software, *Expression Data Clustering Analysis* and *VisualizATiOn Resource* (EXCAVATOR), for testing on three datasets with encouraging results.

To obtain the best possible results, different clustering problems may need different objective functions [31]. Within the MST framework, Xu *et al.* have discussed three different objective functions, and their clustering algorithms partition a tree into $K$ subtrees, where $K$ is any integer greater than zero. The first objective function is to partition an MST into $K$ subtrees so that the total edge-distance of all $K$ subtrees is minimized, and it captures the perception that two data points with a short edge-distance should belong to different clusters and hence be cut. The second algorithm, attempts to partition the MST into $K$ subtrees so that the total distance between the center of each cluster and its data points are minimized. In the third algorithm, data points around the best representatives from the dataset were grouped together. Here, the best representatives are not preselected. Instead, they are the result of the optimization process that attempts to partition the tree into $K$ subtrees. The $K$ representatives are selected simultaneously in such a way that they optimize the objective function.

The results obtained from all the three clustering algorithms were virtually identical for the gene expression data in the budding yeast, *Sacchromyces cerevisiae*. Both the Euclidean distance and the correlation distance were used as distance measures. The program determined that a four-way clustering gives the most natural number of clusters for this problem, and these results are concordant with the annotated analysis outcomes of Eisen *et al.* [32].

### 10.2.6 Protein Structural Domain Decomposition

With the number of protein structures in the *Protein Data Bank* (PDB) increasing at an exponential rate, there is an urgent need to develop reliable and efficient methods for identifying the underlying protein structural domains in each structure. *DomainParser* has been developed by Xu *et al.* [33] to solve this domain decomposition problem. Each residue of a protein is represented as a node and the relationship between residues as edges with a nonnegative capacity depending on the type of the contact, thereby reducing this to a network flow problem. Any edge that has a zero capacity corresponds to a nonexistent edge. The capacity of an edge is defined to reflect the packing between the two residues under consideration.

By finding a minimum cut of the network and minimizing the total cross-edge capacity using the classical Ford–Fulkerson algorithm, Xu *et al.* could solve the two-domain decomposition problem, and by iteratively solving a sequence of two-domain problems, a multidomain decomposition problem was worked out. This algorithm requires a source and a sink, and to avoid a trivial and incorrect partition, two groups of nodes are selected as the source and the sink rather than choosing two nodes from the network representation of the protein as the source and the sink. At the outset, all interesting source–sink networks are listed for a given network. Subsequently, for each source–sink network, Xu *et al.* have enumerated all the minimum source–sink cuts by applying the Picard–Queyranne algorithm [33] based on the residual network of the Ford–Fulkerson algorithm. Different partitions are calculated and ranked using global parameters in the post–processing step.

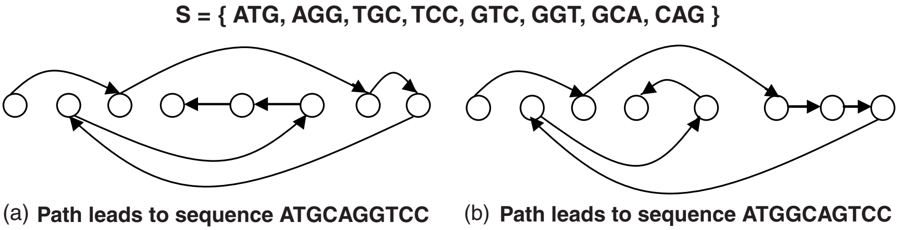
The *DomainParser* program has been tested on a commonly used test set of 55 multidomain protein structures. The decomposition results are 72% in agreement with the literature on both the number of decomposed domains and the assignments of residues to each domain, which compare favorably with other existing programs. On the subset of two-domain proteins, the program assigned 96.7% of the residues correctly.

### 10.2.7 Optimal Design of Thermally Stable Proteins

Making a protein more structurally and functionally stable at higher temperatures makes it advantageous for many industrial and laboratory settings [34]. A more stable protein permits higher process temperatures, which can reduce the risk of microbial contamination and increase both reaction rates and reactant solubility [35, 36]. Mutating specific amino acids to improve explicitly properties of the protein structure by rational or structure-based design is one among the many approaches that have been developed [34]. Directed evolution, which endeavors to accelerate natural evolution in a laboratory setting, is another major innovation in this direction. A computational approach in which data from sequence databases are used is yet another method for making more thermally stable proteins [37, 38].

Bae *et al.* [39] used measures of *local structural entropy* (LSE) to design more thermally stable proteins. They called this novel approach *improved configurational entropy* (ICE). Chan *et al.* [38] examined structures deposited in the *Protein Data Bank* (PDB) to see how often particular amino acid tetramers (four consecutive amino acids from the protein sequence) appeared in protein secondary structures to derive a value for LSE, with a higher LSE value attributed to a tetramer occurring in several kinds of structures compared with one that is restricted to a single secondary structure. Chan *et al.* [38] observed a correlation between LSE and previously published differences of thermal stability for thermophilic proteins and their mesophilc homologues. Bae *et al.* [39] incorporated LSE as part of ICE and used it successfully to select amino acids from closely homologous proteins that minimized the total structural entropy for the target sequence. To solve efficiently the LSE minimization problem, Bannen *et al.* [34] modeled it as a shortest path network optimization problem. A protein sequence of length $n$ is decomposed into an ordered sequence of nodes in a network to construct a network representation. A tetramer was used to label each node with directed edges connecting the nodes, thereby representing each sequence of length $n$ with a path of length $n - 3$. Each node label also was prefixed with its "stage," which is the position of its first amino acid in the protein sequence. For example, the protein sequence MERLTG can be represented as *Source*→(1,M,E,R,L)→(2,E,R,L,T)→(3,R,L,T,G)→*Sink*. The entropy values derived by Chan *et al.* [38] that correspond to the tetramer at the destination represents the values for each edge. The source node with weights equal to the entropies for the tetramers of all the first-stage nodes and a sink node are two other important nodes in this network. A protein sequence of the same length as the original sequence can be arrived at, by finding a path through the network from source to sink [34]. The amino acid at each position is chosen from position-specific amino acid conservation in a multiple sequence alignment of homologous sequences. By finding the shortest path through this network from source to sink, Bannen *et al.* were able to find the optimal sequence using Dijkstra's algorithm to solve this shortest path problem. The optimal string is retrieved from the shortest path by stripping out the node stage numbers and merging the tetramers along the path, thereby eliminating overlaps.

Compared with a simple brute force approach, the shortest path problem using Dijkstra's algorithm improves computational efficiency [34]. With two sequences that are 67% identical and 4800 amino acids long, the shortest path approach generates a graph about the same size as two sequences that are 400 amino acids long and share 0% sequence identity (~6500 nodes) and the shortest path algorithm calculates



(a) **Path leads to sequence ATGCAGGTCC** (b) **Path leads to sequence ATGGCAGTCC**

**Figure 10.19** Using Hamiltonian paths to reconstruct sequences. Different reconstructions for the same given spectrum *S* lead to the sequences (a) ATGCAGGTCC and (b) ATGGCAGTCC.

the overall minimum in less than a second [34]. The overall minimum is found in 26 hours for a graph with ~2 million nodes [34], proving the scalability of the algorithm even in extreme situations.

### 10.2.8 The Sequencing by Hybridization (SBH) Problem

Although a DNA array provides information regarding all strings of length $t$ for an unknown gene sequence, it does not provide information about their positions in the sequence [40]. The $t$-mer composition, or the spectrum, Spectrum($s$, $t$) for a string $s$ of length $n$, is the multiset of $(n - t + 1)$ $t$-mers in s [40]. If $t = 3$ and $s =$ TATGGTGC, then Spectrum($s$,$t$) = {TAT, ATG, TGG, GGT, GTG, TGC}. The primary obstacle of applying DNA arrays for sequencing is the inaccuracy in interpreting hybridization data to distinguish between perfect matches and highly stable mismatches [40].

This *Sequencing By Hybridization* (SBH) problem can be reduced to a Hamiltonian path problem. To construct the graph, a vertex is introduced for every $t$-mer in the Spectrum($s$, $t$), and two vertices are connected by a directed edge if those two vertices overlap [40]. There is a one-to-one correspondence between paths that visit each vertex of $H$ exactly once and DNA fragments represented by the Spectrum($s$, $t$). Oftentimes, it is possible to find more than one Hamiltonian path for a given spectrum, and each of these paths corresponds to different reconstructions. Figure 10.19 shows two such possible paths through a given spectrum. It is clearly shown that the reconstructed sequences are different in both cases. This approach is therefore not very practical when the overlap graph is large.

Another approach is to reduce the SBH problem to an Eulerian path problem [37] in which every edge is visited exactly once. In this new graph, the edges represent the respective $t$-mers from the Spectrum($s$, $t$). A graph is built on all ($t$-1)-mers, rather than on the set of all t-mers. A *(t-1)-mer* node $v$ is joined by a directed edge with a ($t$-1)-mer node $w$, if the spectrum contains a *t-mer* where $v$ is the prefix and $w$ is the suffix of that respective $t$-mer [37]. For a given spectrum, Spectrum($s$, $t$) = {ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}, ($t$-1)-mers and the directed edges between them are shown in Figure 10.20. Figure 10.20a is the graph representation of this spectrum. The sequence can be reconstructed from this graph by finding an Eulerian path. The two possible Eulerian paths through the graph (Figure 10.20a)

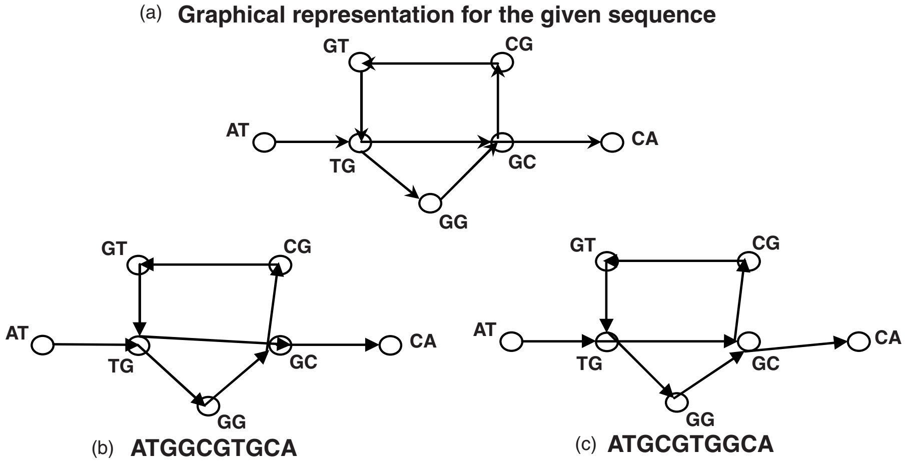


**Figure 10.20** *Sequencing by hybridization* as an Eulerian path problem. (a) Graph representation of the spectrum $S = \{ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT\}$. Two nodes *AT* and *TG* are connected with a directed edge from *AT* to *TG* if there is a 3-mer such that its prefix is *AT* and its suffix is *TG*. Eulerian paths though the graph (a) producing the sequence (b) ATGGCTGCA and (c) ATGCGTGGCA.

are presented (Figure 10.20b and Figure 10.20c). Although finding an Eulerian path may seem as difficult as finding a Hamiltonian path, Jones and Pevzner [40] have stated that finding an Eulerian path can be implemented in time linear in the number of edges in the graph. Furthermore, they have suggested that in a balanced graph, it is possible to find more than one Eulerian path and then combine these different paths into one to obtain the single path.

### 10.2.9 Predicting Interactions in Protein Networks by Completing Defective Cliques

Identification of the complete set of interactions among proteins in a cell is a fundamental problem in modern biology [41,42]. The experimental techniques available are either small-scale or large-scale experimental methods. For a given set of proteins, small-scale techniques such as coimmunoprecipitation recognize the interactions between a pair of proteins at a time [43–46]. Large-scale techniques like yeast two-hybrid and tandem affinity purification (TAP) tagging, on the other hand, locate numerous interacting pairs in a single experiment [47–50].

Yu *et al.* [9] propose using a graph-based method to represent a protein interaction network. Proteins are represented as vertices and pairs of interacting proteins are connected by edges. The matrix model interpretation of the results of these large-scale experiments reveals that two proteins interacting with the same protein cluster are likely to interact with each other, leading to a predicted interaction [9]. Using defective cliques to carry forward their algorithm, they observed that if nonadjacent vertices $P$ and $Q$ are both adjacent to every vertex in a clique $K$, then it is likely that

$P$ and $Q$ are adjacent to each other [9]. $P$, $Q$, and $K$ then form a defective clique. Their algorithm searches for these special defective cliques in protein interaction graphs and predicts interactions that complete these defective cliques, resulting in full cliques.

The algorithm first identifies all cliques in a network and finds pairs of cliques overlapping on all but one node each. Then, edges between the overlapping and the nonoverlapping nodes, in each of these pairs are predicted, and the new node is added to the network. The efficiency of the algorithm is improved by looking for partial overlaps of maximal cliques (a clique that is not contained in any other clique in the graph) that differ in more than one node.

Yu *et al.* [9] applied their clique completion method to a large-scale experimental dataset of the protein interaction network of *S. cerevisiae* obtained by Bader and Hogue [51], with excellent results.

## 10.3 CONCLUSION

Graph theory has a wide range of applications in bioinformatics, represented by DNA fragment assembly, protein domain decomposition, evolutionary tree construction, visualization techniques for alternative splicing, identification of structural domains, and clustering sequence alignments. In light of the many methodologies described in this chapter, it is evident that the key algorithms of graph theory can be applied successfully to address various biological problems. With the advent of a systems biology approach to deciphering biological processes, we expect graph theoretical methods to be applied extensively in the future.

## REFERENCES

1. N. Biggs, E. Lloyd, and R. Wilson. Graph Theory. Oxford University Press, New York, 1986.
2. Graph theory. Encyclopedia Britannica Online, 2009. http://www.britannica.com/EBchecked/topic/242012/graph-theory.
3. J.H. Barnett. Early Writing on Graph Theory: Hamiltonian Circuits and The Icosian Game. http://www.math.nmsu.edu/hist_projects/Hamilton-Barnett.pdf.
4. J.J. Sylvester. Chemistry and algebra. Nature, 17:284, 1878.
5. A. Doerr and K. Levasseur. Applied Discrete Structures for Computer Science. Galgotia Publications Pvt. Ltd., New Delhi, India, 2000.
6. J. Gross and J. Yellen. Graph Theory and Its Applications. CRC Press, Boca Raton, FL, 2000.
7. K. Thulasiraman and M.N.S. Swamy. Graphs: Theory and Algorithms. Wiley, New York, 1992.
8. W.T. Tutte and C.S.J.A. Nash-Williams. Graph Theory. Cambridge University Press, New York, 2001.
9. H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006.

10. C. Korostensky and G.H. Gonnet. Using traveling salesman problem algorithms for evolutionary tree construction. *Bioinformatics*, 16(7):619–627, 2000.
11. J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc Am Math Soc*, 7(1):48–50, 1956.
12. R. Campos and M. Ricardo. A fast algorithm for finding minimum routing cost spanning trees. *Comput Network*, 52(17):3229–3247, 2008.
13. D.P. Bertsekas. A simple and fast label correcting algorithm for shortest paths. *Networks*, 23(7):703–709, 1992.
14. E.W. Dijkstra. A note on two problems in connexion with graphs. *Numer Math*, 1:269–271, 1959.
15. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introductions to algorithms. MIT Press and McGraw-Hill, Boston, MA, 2001.
16. L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Can J Math*, 8:399–404, 1956.
17. L. Eugene. Combinatorial Optimization: Networks and Matroids. *Oxford University Press*, New York, 1976.
18. L.D. Stein. Integrating biological databases. *Nat Rev Genet*, 4(5):337–345, 2003.
19. N.M. Luscombe, D. Greenbaum, and M. Gerstein. What is bioinformatics? an introduction and overview. *Yearbook of Medical Informatics*, 2001.
20. W. Huber, V.J. Carey, L. Long, S. Falcon, and R. Gentleman. Graphs in molecular biology. *BMC Bioinformatics*, 8(6):S8, 2007.
21. D. Bollina, B.T.K. Lee, T.W. Tan, and S. Ranganathan. ASGS: An alternative splicing graph web service. *Nucleic Acids Res*, 34:W444–W447, 2006.
22. M. Krawczak, J. Reiss, and D.N. Cooper. The mutational spectrum of single base-pair substitutions in mRNA splice junctions of human genes: causes and consequences. *Hum Genet*, 90(1–2):41–54, 1992.
23. B.T.K. Lee, T.W. Tan, and S. Ranganathan. DEDB: a database of *Drosophila melanogaster* exons in splicing graph form. *BMC Bioinformatics*, 5:189, 2004.
24. W.M. Fitch and E. Margoliash. The construction of phylogenetic trees. *Science*, 155(760):279–284, 1967.
25. G.H. Gonnet and S.A. Benner. Probabilistic ancestral sequences and multiple alignments. *Fifth Scandinavian Workshop on Algorithm Theory*, Reykjavik, Iceland, pp. 380–391, 1996.
26. P. Magwene, P. Lizardi, and J. Kim. Reconstructing the temporal ordering of biological samples using microarray data. *Bioinformatics*, 19(7):842–850, 2002.
27. J.S. Richardson. The anatomy and taxonomy of protein structure. *Adv Protein Chem*, 34:167–339, 1981.
28. D.B. Wetlaufer. Nucleation, rapid folding, and globular intrachain regions in proteins. *Proc Natl Acad Sci U S A*, 70(3):697–701, 1973.
29. H. Hegyi and M. Gerstein. The relationship between protein structure and function: a comprehensive survey with application to the yeast genome. *J Mol Biol*, 288(1):147–164, 1999.
30. X. Guan and L. Du. Domain identification by clustering sequence alignments. *Bioinformatics*, 14:783–788, 1998.
31. Y. Xu, V. Olman, and D. Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning tree. *Bioinformatics*, 18(4):536–545, 2002.

32. M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Bostein. Cluster analysis and display of genome-wide expression pattern. *Proc Natl Acad Sci U S A*, 95:14863–14868, 1998.

33. Y. Xu, D. Xu, and H.N. Gabow. Protein domain decomposition using a graph-theoretic approach. *Bioinformatics*, 17(3):1091–1104, 2000.

34. R.M. Bannen, V. Suresh, G.N. Phillips Jr. S.J. Wright, and J.C. Mitchell. Optimal design of thermally stable proteins. *Bioinformatics*, 24(20):2339–2343, 2008.

35. V.G. Eijsink, A. Bjørk, S. Gåseidnes, R. Sirevåg, B. Synstad, B. van den Burg, and G. Vriend. Rational engineering of enzyme stability. *J Biotechnol*, 113(1–3):105–120, 2004.

36. P. Turner, G. Mamo, and E.N. Karlsson. Potential and utilization of thermophiles and thermostable enzymes in biorefining. *Microb Cell Fact*, 6:9, 2007.

37. M. Lehmann and M. Wyss. Engineering proteins for thermostability: The use of sequence alignments versus rational design and directed evolution. *Curr Opin Biotechnol*, 12(4):371–375, 2001.

38. C.H. Chan, H.K. Liang, N.W. Hsiao, M.T. Ko, P.C. Lyu, and J.K. Hwang. Relationship between local structural entropy and protein thermostability. *Proteins*, 57(4):684–691, 2004.

39. E. Bae, R.M. Bannen, and G.N. Phillips Jr. Bioinformatic method for protein thermal stabilization by structural entropy optimization. *Proc Natl Acad Sci U S A*, 105(28):9594–9597, 2008.

40. N.C. Jones and P.A. Pevzner. Introduction to bioinformatics algorithms. *MIT Press*, Cambridge, MA, 2004.

41. R. Jansen, H. Yu, D. Greenbaum, Y. Kluger, N.J. Krogan, S. Chung, A. Emili, M. Snyder, J.F. Greenblatt, and M. Gerstein. A Bayesian networks approach for predicting protein–protein interactions from genomic data. *Science*, 302(5644), 449–453, 2003.

42. E.M. Marcotte, M. Pellegrini, H.L. Ng, D.W. Rice, and T.O. Yeates. Detecting protein function and protein–protein interactions from genome sequences. *Science*, 285(5438):751–753, 1999.

43. G.D. Bader, I. Donaldson, C. Wolting, B.F. Ouellette, T. Pawson, and C.W. Hogue. BIND–the biomolecular interaction network database. *Nucleic Acids Res*, 29(1):242–245, 2001.

44. H.W. Mewes, D. Frishman, U. Guldener, G. Mannhaupt, K. Mayer, M. Mokrejs, B. Morgenstern, M. Münsterkötter, S. Rudd, and B. Weil. MIPS: A database for genomes and protein sequences. *Nucleic Acids Res*, 30(1):31–34, 2002.

45. I. Xenarios, L. Salwínski, X.J. Duan, P. Higney, S.M. Kim, and D. Eisenberg. DIP, the database of interacting proteins: A research tool for studying cellular networks of protein interactions. *Nucleic Acids Res*, 30(1):303–305, 2002.

46. Y. Xia, H. Yu, R. Jansen, M. Seringhaus, S. Baxter, D. Greenbaum, H. Zhao, and M. Gerstein. Analyzing cellular biochemistry in terms of molecular networks. *Annu Rev Biochem*, 73:1051–1087, 2004.

47. A.C. Gavin, M. Bösche, R. Krause, P. Grandi, M. Marzioch, A. Bauer, J. Schultz, J.M. Rick, A.M. Michon, C.M. Cruciat, M. Remor, C. Höfert, M. Schelder, M. Brajenovic, H. Ruffner, A. Merino, K. Klein, M. Hudak, D. Dickson, T. Rudi, V. Gnau, A. Bauch, S. Bastuck, B. Huhse, C. Leutwein, M.A. Heurtier, R.R. Copley, A. Edelmann, E. Querfurth, V. Rybin, G. Drewes, M. Raida, T. Bouwmeester, P. Bork, B. Seraphin, B. Kuster, G. Neubauer, and G. Superti-Furga. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–147, 2002.

48. Y. Ho, A. Gruhler, A. Heilbut, G.D. Bader, L. Moore, S.L. Adams, A. Millar, P. Taylor, K. Bennett, K. Boutilier, L. Yang, C. Wolting, I. Donaldson, S. Schandorff, J. Shewnarane, M. Vo, J. Taggart, M. Goudreault, B. Muskat, C. Alfarano, D. Dewar, Z. Lin, K. Michalickova, A.R. Willems, H. Sassi, P.A. Nielsen, K.J. Rasmussen, J.R. Andersen, L.E. Johansen, L.H. Hansen, H. Jespersen, A. Podtelejnikov, E. Nielsen, J. Crawford, V. Poulsen, B.D. Sørensen, J. Matthiesen, R.C. Hendrickson, F. Gleeson, T. Pawson, M.F. Moran, D. Durocher, M. Mann, C.W. Hogue, D. Figeys, and M. Tyers. Systematic identification of protein complexes in *Saccharomyces cerevisiae* by mass spectrometry. *Nature*, 415(6868):180–183, 2002.

49. T. Ito, K. Tashiro, S. Muta, R. Ozawa, T. Chiba, M. Nishizawa, K. Yamamoto, S. Kuhara, and Y. Sakaki. Toward a protein–protein interaction map of the budding yeast: A comprehensive system to examine two-hybrid interactions in all possible combinations between the yeast proteins. *Proc Natl Acad Sci U S A*, 97(3):1143–1147, 2000.

50. P. Uetz, L. Giot, G. Cagney, T.A. Mansfield, R.S. Judson, J.R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J.M. Rothberg. A comprehensive analysis of protein–protein interactions in *Saccharomyces cerevisiae. Nature*, 403, 623–627, 2000.

51. G.D. Bader and C.W. Hogue. Analyzing yeast protein–protein interaction data obtained from different sources. *Nat Biotechnol*, 20(10), 991–997, 2002.

# 11

# A FLEXIBLE DATA STORE FOR MANAGING BIOINFORMATICS DATA

Bassam A. Alqaralleh, Chen Wang, Bing Bing Zhou, and Albert Y. Zomaya

## 11.1 INTRODUCTION

With the abundance of scientific data in recent years, how to manage them effectively becomes a challenging problem. Data produced by scientific activities often evolve quickly and are too dynamic to have broadly agreed metadata structures. Bioinformatics data belong to this category. The advances of high-throughput genome sequencing and gene expression profiling technologies produce huge amounts of data. They are open for new interpretations, and the interpretations may change when new discoveries are made. The research community use data annotation heavily to record these interpretations. In a certain time frame, there could be a burst of annotations on certain data. Traditional database systems do not provide enough flexibility in managing such kinds of data. There are efforts to build new data management systems. A promising one is to allow data to be stored freely in any format and to index these data using small pieces of structured information so that data can be retrieved through these indexes. This technique has been attempted in social networking sites like flickr and del.icio.us, which allows users to annotate pictures and bookmarks flexibly. The structures of these annotations are as simple as tags and key-value pairs. The simplicity has advantage in usability but raises challenges for the accuracy of data search. It is therefore not sufficient for indexing bioinformatics data. In this chapter, we investigate the use of a multidimensional indexed data store to tackle this problem.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Multidimensional indexing is important to a variety of data-centric applications, ranging from sequence matching, to protein structure matching, and to spatial databases [19]. In a multidimensional indexing scheme, a data element is indexed using multiple keys selected from the data element's attribute list. A query containing the indexed keys therefore can locate the target data element quickly via the index.

### 11.1.1 Background

Multidimensional indexing and spatial query processing have been studied extensively in centralized systems. In many of these methods, the data space is divided hierarchicaly into smaller subspaces (regions), such that the higher level data space contains the lower level subspaces and acts as a guide in searching. Among these various spatial index structures [3, 11], R-tree [8] and its variations (R*-tree [2], R+-tree [13], etc.) are used widely in the industry and research communities. The R-tree is a multidimensional extension of the B+-tree. In R-trees, each spatial data object is represented by a minimum bounding rectangle (MBR), which is used to store the leaf node data entries.

R-trees efficiently can answer various types of multidimensional queries, especially range query. Given a query window $q$, a range query retrieves all objects inside or intersecting $q$. Range query answering starts from the root level of the R-tree. For any MBR intersecting the query window, its subtrees are explored recursively. If a leaf entry is encountered, then all objects whose bounding range intersects the query window are examined [8]. Figure 11.1 shows two-dimensional space with some data objects, and Figure 11.2 shows its corresponding R-tree.

### 11.1.2 Scalability Challenges

The scalability of a data management system includes two aspects: scaling up incrementally as data grow in size and scaling up incrementally as demands for these data and data indexes grow.

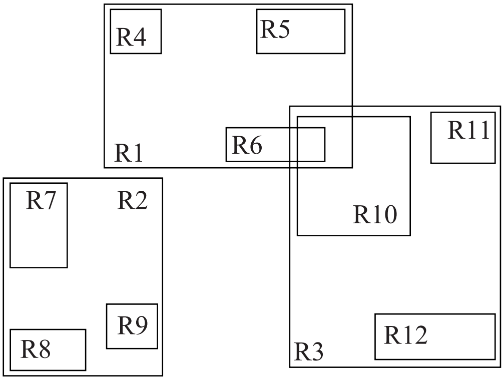


**Figure 11.1** Data rectangles.

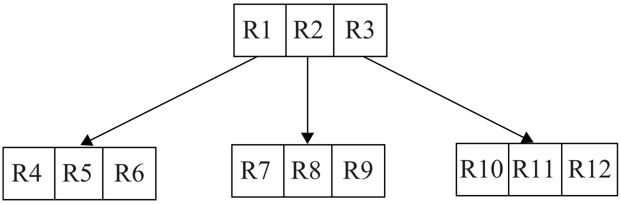


**Figure 11.2** R-tree.

Recently, Internet companies, such as Google (Mountain View, CA) and Amazon (Seattle, WA) built large-scale data centers for scaling up their applications. These data centers often consist of tens of thousands of commodity personal computers (PCs). Their data management systems can deal with structured data (e.g., Amazon system has a key-value store (Dynamo) [20], and Google cluster supports a sparse sorted map (Bigtable) [21]). Data with simple structures are convenient to partition, and these systems often scale up data storage through partitioning data and storing partitions incrementally in available nodes. However, they leave the access scalability to applications to handle. The replication mechanisms in these systems often only serve for high availability and fault-tolerance purpose. When a dataset is requested heavily, the application using the dataset has to deal with the scalability itself. It is difficult for individual applications to handle this when the dataset is shared. Existing systems lack of a systematic approach for addressing this issue.

In this chapter, we propose a data model that allows multidimensional data indexing in a data store. The data model is flexible and expressive for bioinformatics data annotation and indexing. It is also convenient to build a scalable data store to support this data model. We detail the mechanisms used for constructing a self-organizing data store in this chapter. The mechanism is scalable for both data storage and access request processing.

The rest of the chapter is organized as follows: Section 11.2 describes the data model and the system architecture supporting this data model; Section 11.3 introduces the algorithms for scaling up the system in a self-organized manner; Section 11.4 presents the simulation results; Section 11.5 summarizes related work, and Section 11.6 summarizes the chapter.

## 11.2 DATA MODEL AND SYSTEM OVERVIEW

In our system, a structured data item is represented as a set of tuples in the format (attribute, type, value). For example, a protein sequence can be represented as follows:

$$\{(\text{gene\_bank\_ID, string, ``gi : 1000344''}),$$
$$(\text{db\_source, string, ``locus CEU34596 accession U34596.1''}),$$
$$(\text{length, int, 570})\}.$$

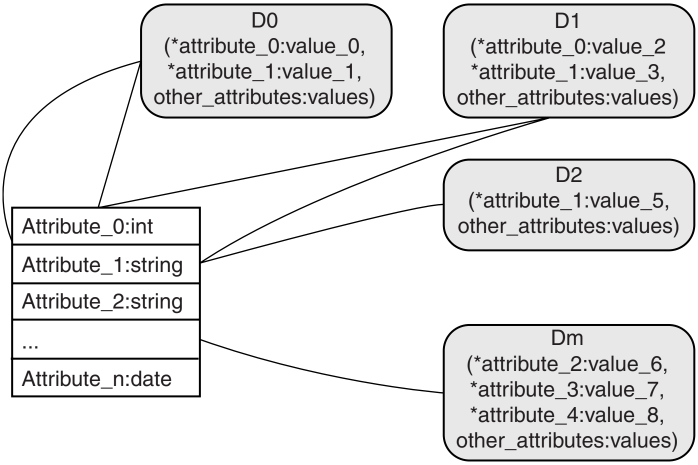


**Figure 11.3** An example of bio-data representation using the proposed data model.

We assume all attributes are defined in the same namespace. A subset of attributes can be used to index data items (*e.g.*, gene_bank_ID can be used to keep track the location where the data item is stored). This index can be recorded in the following form:

$$(\text{gi:1000344, node\_x:uid})$$

in which uid is the unique key corresponding to the data item. The unique key is generated using a collision resilient hash function like MD5 or SHA-1. When a data item is indexed using more than one attribute, it becomes a multidimensional indexed data item.

Data items often have an overlapped attribute set and are indexed using a common subset of attributes. Figure 11.3 shows such an example. The table on the left is the attribute set used by all data items and $D0$, $D1$, and $Dm$ denote different data items. $D0$ and $D1$ are indexed using ($attribute\_0$, $attribute\_1$) (indexing attributes are marked with "*"). $D2$ is indexed using ($attribute\_1$), and $Dm$ is indexed using ($attribute\_2$, $attribute\_3$, and $attribute\_4$). Note, $D2$ also can be treated as multidimensional indexed by adding unconstrained attributes to it (*e.g.*, it can share the same index repository with $D0$ and $D1$ by allowing $attribute\_0$ to anything).

In general, the same indexing attribute set forms an index repository. The repository can be identified by hashing the concatenated attribute set using the same collision resilient hash function mentioned. The output of the hash function is used to map the repository to a physical node. The mapping is done using a distributed hash table (DHT) algorithm Pastry [14] in our system.

Figure 11.4 shows our system architecture. Based on the role a node may play in answering a client query, the nodes in our system can be classified into three categories: routing nodes, index nodes, and data nodes.

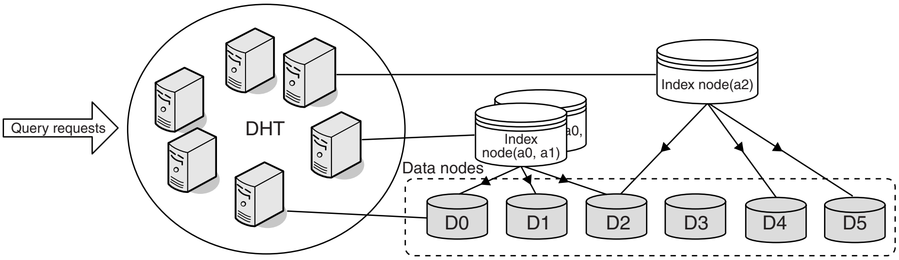


**Figure 11.4** The system architecture (index nodes and data nodes are part of the DHT overlay).

Each node in the system can be a routing node. A node forwards messages according to the Pastry algorithm.

An index node stores the multidimensional indexes (R-trees) of a certain type of data. An index node is addressable in the DHT network via the hash value of the concatenated keys in its index, for example, the following query:

$$\text{Q1} = (\text{select * from Overlay where } a_0 = x \text{ and } a_1 = y)$$

will be directed to a node whose identification (ID) is closest to $h(a_0||a_1)$, where $h$ is a collision resilient function and $||$ is concatenation.

A data node stores the actual data. A piece of data has an ID generated from its content, and this ID is used to map the data to a target storage node. To maintain the data locality, the ID of the tuple is generated using a locality sensitive hash function so that similar data are stored close to each other in the DHT overlay.

To insert a data element $(a_0 : v_0, a_1 : v_1, ..., a_n : v_n)$ indexed using an attribute set $(a_i, a_{i+1}, ..., a_{i+m})$ to the system, we may use an inserting statement in the following syntax:

$$\text{insert } (a_0 : v_0, a_1 : v_1, \ldots, a_n : v_n) \text{ indexedBy } (a_i, a_{i+1}, \ldots, a_{i+m})$$

The insertion process is described as follows:

- Step 1
  A client submits the insertion request to any node in the overlay network.
- Step 2
  The DHT node that receives the request, denoted by $N_e$, calculates the data ID using a locality sensitive hash function (LSH) as follows: $d = \text{lsh}(a_i||a_{i+1}||\ldots||a_{i+m}||v_i||v_{i+1}||\ldots||v_{i+m})$.
- Step 3
  $N_e$ sends a data insertion request to the data node with an ID closest to $d$. If the request is accepted, then the data is sent to that data node, denoted by $N_d$.
- Step 4
  If the request is rejected, then $N_e$ recalculates the data ID as $d = \text{lsh}(a_i||a_{i+1}||...||a_{i+m}||v_i||v_{i+1}||...||v_{i+m}||r)$, where $r$ is a random number. $N_e$

repeats Step 3 untill the request is accepted by a data node. A data node rejects the data insertion request mainly because of its storage capacity. If no data node accepts the request after a predefined maximum number of attempts, then Ne declares the insertion failed and returns an "out of space" error message to the client.

- STEP 5
  $N_e$ locates the index node by calculating the hash ID from the index keys: $I = h(a_i||a_{i+1}||\ldots||a_{i+m})$, where $h$ is a collision resilient hash function, and then forwards an indexing request $(I, (a_i : v_i, a_{i+1} : v_{i+1}, \ldots, a_{i+m} : v_{i+m}), N_d)$ to the index node $N_I$.
- STEP 6
  $N_I$ inserts the data index to its R-tree and notifies $N_e$. A data element also can be indexed by multiple indexes, such as,

  $$\text{insert } (a_0 : v_0, a_1 : v_1, \ldots, a_n : v_n) \text{ indexedBy } (a_i, a_{i+1}, \ldots, a_{i+p}),$$
  $$(a_j, a_{j+1}, \ldots, a_{j+q})$$

  Step 5 and 6 repeat if multiple indexes present in the insert statement.
- STEP 7
  $N_e$ returns success to the client.

The indexing mechanism supports two types of queries; one is a point query in which a query statement contains known values of some attributes; the other is a range query in which a query statement specifies the value range of some attributes. In both cases, a query is submitted to any DHT node and is forwarded to an index node based on the attributes in the query. The index node returns the IDs of qualified data and their locations (*i.e.*, the data node IDs). The DHT node then gets data from those data nodes, assembles the results for range queries, and returns the results to the client. The process is illustrated in Figure 11.5.

Each index/data node maintains a first come first serve (FCFS) queue for incoming query/data-fetch requests. The queue is lossless, as it does not drop queuing

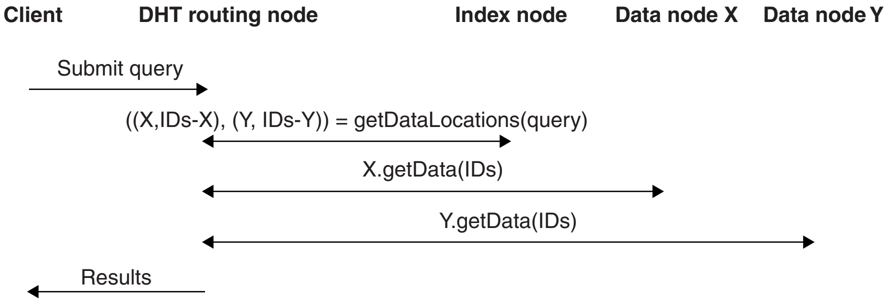


**Figure 11.5** The query resolving process.

requests. To ensure quality of service, a queue is associated with a predefined value called capacity. The capacity is defined as the number of queries the node can process in a certain time frame. When the capacity is reached, the node is considered overloaded, and the requests coming subsequently may suffer a long delay. To reduce the queue length, requests may be forwarded to a node that can serve the request. When such a node cannot be found, our system replicates the content of the overloaded node to a lightly loaded node. The query serving capacity increases once the new replica is created, and therefore, the load on the overloaded node will decrease gradually.

## 11.3 REPLICATION AND LOAD BALANCING

The data in our system can be diverse, and the access to these data is dynamic. It is difficult to allocate nodes for data storage in a static manner. When demands increase, both index nodes and data nodes in our system can expand themselves in a self-organized manner. This is done through replication by dynamically recruiting nodes in the overlay to host their replicas.

Nodes storing the same data form a content distribution network for the data. In its simplest form, the content distribution network of a data ID is fully connected, and load information is exchanged periodically along these links so that the forwarding destination can be obtained easily. When the data hosted by a node is popular, it is likely that the queue size is close to or over the capacity most of the time. When all nodes in a content distribution network are overloaded, the network will be expanded by creating new replicas. The new replica node is selected from available nodes in the underlying network. Each node in the system autonomously can create replicas of its local data objects. The whole system is organized in a decentralized manner. Each node makes a replication decision by itself.

Our replication and load balancing mechanism intends to achieve the following goals:

1. To improve query service quality in terms of queuing time, traveling time, and response time
2. To optimize the number of replicas needed for serving queries to a hot content by balancing the load among existing content nodes that host this data item

Replicating and load balancing are implemented on a simple protocol we first introduced in [22]. The protocol is summarized in Figure 11.6. There are three types of messages in the protocol:

1. Messages used by an overloaded content node to recruit new replica node
2. Messages used to forward queries to lightly loaded nodes
3. Messages used by a node to update other nodes with its current workload. This is the information used for load balancing

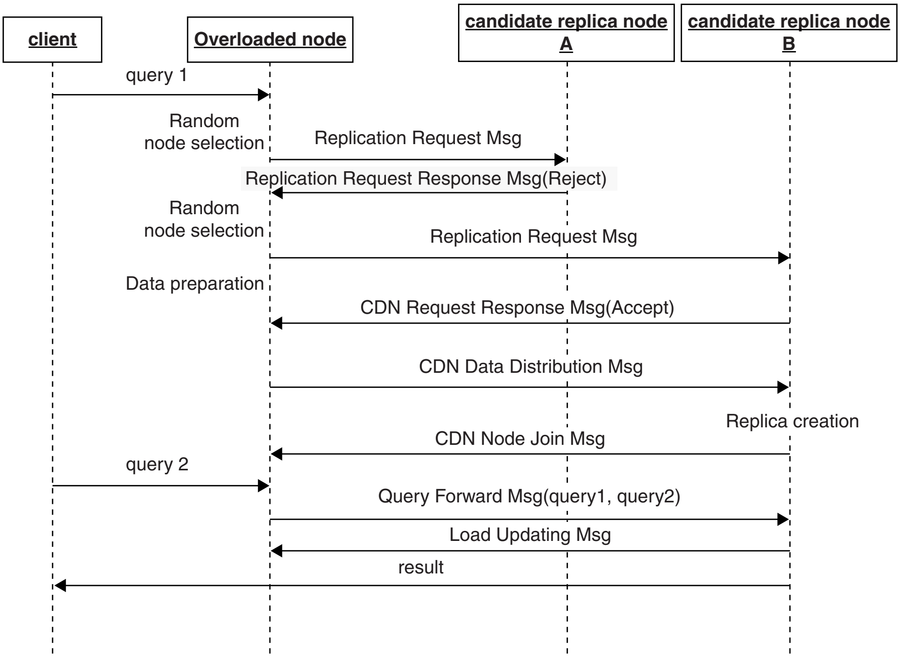


**Figure 11.6** Illustration of replica management protocol.

An overloaded node may send a replication request to multiple candidate nodes simultaneously, but only one that accepts the request is selected at a time to replicate the highly requested data.

### 11.3.1 Replicating an Index Node

As all queries to a certain type of data go through a particular index node, the node can be overwhelmed easily. Replication of an index node includes the following steps:

- Step 1
  Mark the start of snapshot taking for the R-tree of the overloaded node.
- Step 2
  Create a temporary queue to hold the incoming insert/update requests.
- Step 3
  Create the snapshot. During this step, only query requests are processed in the index node.
- Step 4
  After the snapshot is taken, allow the processing of insert/update requests on the index node; meanwhile, replicate the snapshot to a selected lightly-loaded node.

- STEP 5
  Forward the insert/update requests in the temporary queue to the replica node.
- STEP 6
  Delete the temporary queue when it is empty.

After these steps, the new index node is ready to use, and incoming queries can be forwarded to the new node for processing through the load balancing mechanism. The subsequent insert/update requests are forwarded immediately to replica nodes to update the replicas. The mechanism implements a relaxed consistency, and there is a small window that replicas are in different states. Our choice is based on the assumption that the arrival rate of insert/update requests is much lower than that of the query requests. This is realistic in our target problem domain in which queries can tolerate data inconsistency in a short time frame within a predefined freshness bound. However, our replication mechanism can be configured to support strong data consistency when needed. This only requires modifications in Steps 2 and 3 to hold the incoming query requests in the temporary queue before the snapshot is taken. The drawback is the increase of response time of the queries in the temporary queue.

On the other hand, a DHT routing node caches the locations of nodes that answer queries passing through it. By doing so, the subsequent query to the same type of data may be directed to a replica of an indexing node, and it avoids going through the original index node. This further can reduce the workload of the original index node.

To balance the load on replicas, our system requires each replica node to update its queue length to other replica nodes when the change exceeds a predefined percentage. Each replica node therefore can forward requests to the least-loaded node when its own load exceeds a predefined threshold. As our load-updating mechanism does not guarantee the load information fresh enough, there is a chance that a replica node is no longer lightly loaded when a forwarded request arrives. In this case, the request is forwarded further to another lightly loaded node based on the node's local information. A request is forwarded like this until a predefined maximum forwarding hops is reached. The query will be inserted into the request queue in the last node it visits. When this occurs, the whole content distribution network for the data index is likely to be overloaded. The last hop node of the request will elect itself to expand the content distribution network by recruiting new replica node to join. The candidate node is selected randomly from the DHT ID space.

### 11.3.2 Answering Range Queries with Replicas

Range queries seek nodes with data regions that intersect the query region. Because many MBRs (minimum bounding rectangles) may intersect with a given query region, the index node may return the locations of multiple data nodes for answering a range query. As a result, the response time of the query is determined by the slowest data node that stores the data intersecting the data region in the query.

With the self-organized replication mechanism, we effectively can reduce the chance that an overloaded node becomes the bottleneck for answering range queries. As each index/data node creates replica to keep the length of its request queue lower than a predefined threshold, the difference in request queue lengths among data nodes selected to answer a range query is bounded by the threshold of these queues. As a result, it is unlikely that the range query answering takes a long time waiting for a slow data node.

## 11.4 EVALUATION

We run extensive simulations to evaluate our mechanism. Our simulator uses FreePastry to construct the underlying DHT network, which consists of 1000 nodes in total.

We generate 100,000 data objects randomly in multidimensional space (two- and three-dimensional) and publish them on the overlay by indexing data values on these dimensions. The length of each dimension is 1000 units. We randomize the index processing time between 10 ms and 20 ms, and the data processing time between 40 ms and 60 ms. The setting is based on the data access time in common database systems. The query forwarding delay between two nodes is set to 2 ms. The capacity of each node (query queue length), including indexing node and data node, is set to 10.

### 11.4.1 Point Query Processing Performance

We used a two-dimensional space without overlapped regions to investigate the cost of point queries. Point query cost is measured by the number of routing hops and response time. Queries are generated using two different distributions to simulate different access patterns. In the first one, clients search for data points uniform and randomly distributed in data space. The data points in a two-dimensional space are illustrated in Figure 11.7. In the second one, clients search for data points that are

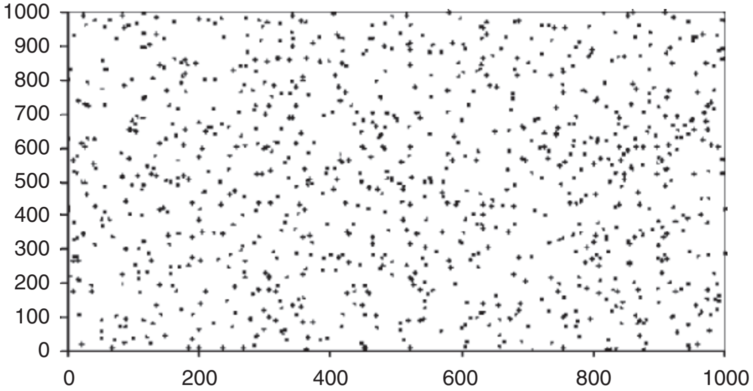


Figure 11.7 Illustration of data points of 1000 queries (uniform distributed).

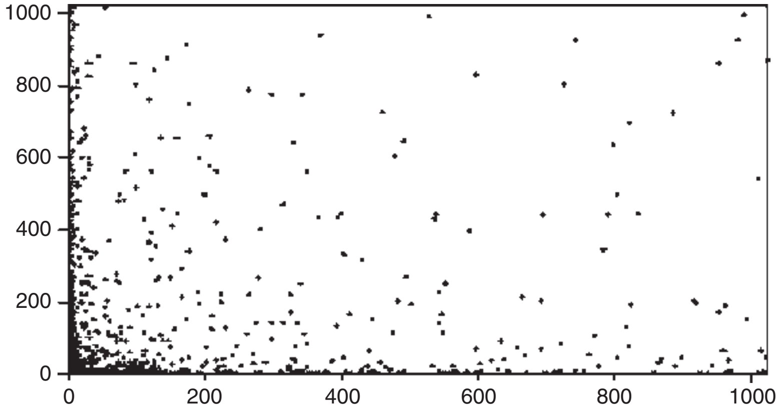


**Figure 11.8** Illustration of data points of 1000 queries (*zipf* distribution with $\alpha = 1.0$).

distributed in the data space according to *Zipfian*, or *zipf* distribution, as shown in Figure 11.8. *Zipf* distribution creates a high access load toward a few data nodes (hotspots). We use 30,000 queries to feed the system. Queries arrive in the system in a Poisson process.

***11.4.1.1 Response Time.*** Figrue 11.9 shows the average query response time under a different query arrival rate. As expected, queries to skewed data points incur a higher average response time than those to uniformly distributed data points. This is because skewed queries easily can overwhelm nodes hosting popular data and therefore trigger the creation of more replicas. The cost of creating additional replicas results in a longer average query response time. One may notice that in

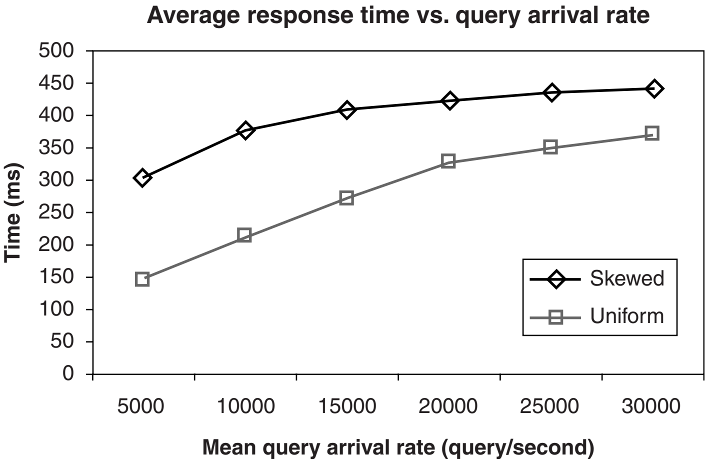


**Figure 11.9** Average query response time under different workload – point query, 30000 queries.

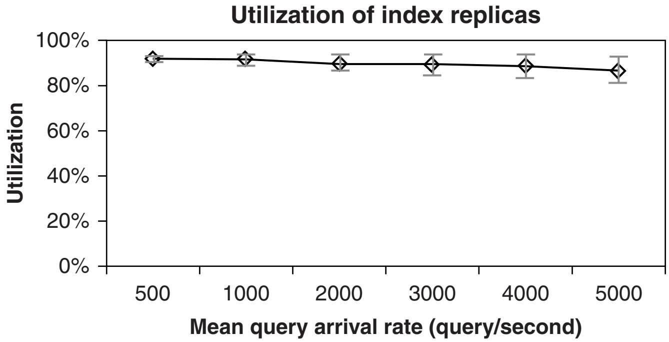


**Figure 11.10** Utilization of index replicas under different workload.

Figure 11.9, the average response time for queries of uniformly and randomly distributed data points keeps increasing, whereas the average response time for queries of skewed data points quickly becomes stable. This can be explained by the number of nodes involved in query answering. When workload is low, queries are answered by more nodes in a “uniform” case than in a “skewed” case. The replica creation speed is slower in “uniform” than in “skewed.” When the workload becomes high, queries overwhelms most data nodes, even for nonskewed queries, and incur increasing replica creation cost.

***11.4.1.2 Node Utilization.*** We measure how load is balanced among replicas by comparing the node utilization of each replica. The node utilization is calculated as the percentage of busy time in the node’s lifetime.

The lifetime of a node is between its creating time and the time when the system finishes processing the last query.

Figure 11.10 shows the average utilization of index content nodes is greater than 80%. When the query arrival rate increases, the average utilization slightly decreases. This is mainly because of the increasing cost of query forwarding and load updating among replica nodes. On the other hand, our self-organizing mechanism can avoid creating unnecessary replicas, as there is no significant decrease in the average node utilization.

Figure 11.11 shows the standard deviation of the replica utilizations of index nodes, which directly reflects how imbalanced the load is among replicas of an index node. As the query arrival rate increases, the difference between replica nodes increases; however, the difference is generally very small.

Figure 11.12 and Figure 11.13 measure the load balancing among replicas of data nodes. The load among replicas of data nodes is not as balanced as that among replicas of the index node. There are two factors contributing to this:

1. The load on a data node is affected by the performance of the index node that dispatches queries to it
2. More data nodes are involved in answering queries in the data layer

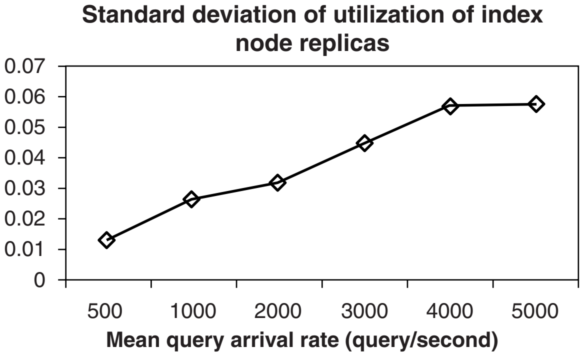


**Figure 11.11** The standard deviation of utilization of replicas of an index node.

### 11.4.2 Range Query Processing Performance

In range query experiments, we compose range queries by selecting the central data points of queries in multidimensional data space according to uniform random distribution. The sizes of query windows (the area has equal edge length) are set to be 0.25%, 1%, and 2% of the total area size.

Our range query experiments are designed to observe how a content distribution network grows and how it performs when it becomes stable. To achieve this, we feed the simulator two groups of queries with queries arriving at the same rate in each group. The first group consists of 10,000 queries. These queries make the content distribution network grow to a size capable of serving queries in a satisfactory response time. The second group also consists of 10,000 queries. These queries are used to measure the query serving quality when the content distribution network is built. The following results are collected during the stable phase.

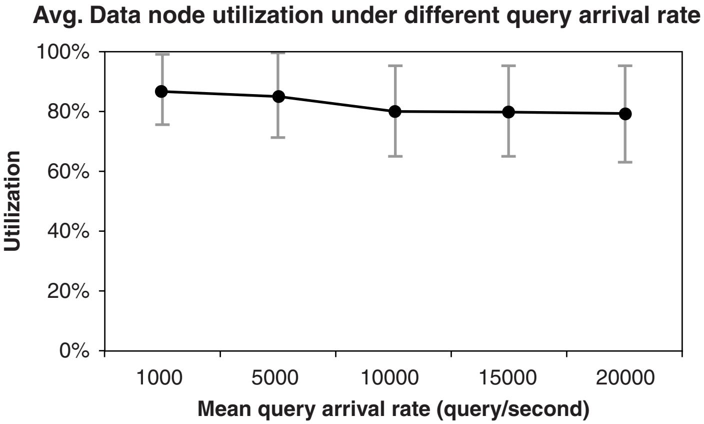


**Figure 11.12** Utilization of data node replicas under different workload: uniformly distributed point queries.

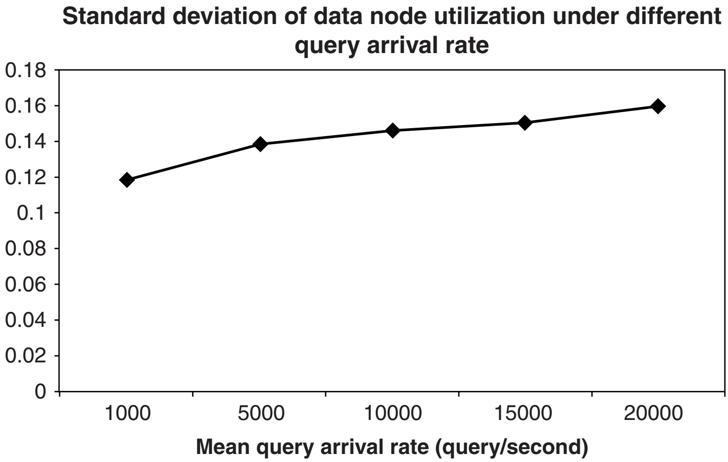


**Figure 11.13** The standard deviation of utilizations of data node replicas: in uniformly distributed point queries.

***11.4.2.1 Response Time.*** Figure 11.14 and 11.15 show the average query response under different query arrival rates and different query window sizes.

We can see that the response time is affected by query window size. This is because a bigger query window overlaps more MBRs in an R-tree, and more nodes therefore are involved in answering a query.

Figures 11.14 and 11.15 also show that query response time is not sensitive to the change of query arrival rate, which reveals the good scalability of our mechanism. One also may find that the dimensionality does not affect the response time much by comparing Figure 11.14 and Figure 11.15.

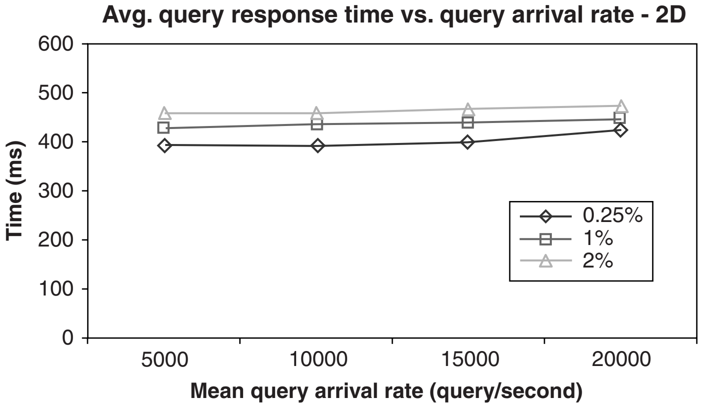


**Figure 11.14** Average query response time for a two-dimensional query with 10,000 queries.

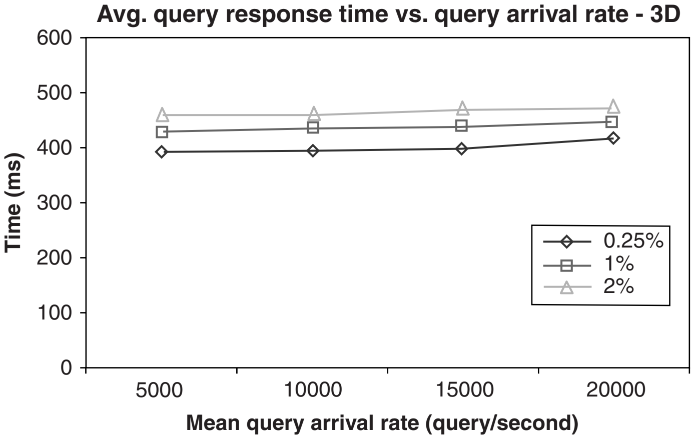


**Figure 11.15** Average query response time for a three-dimensional query with 10,000 queries.

***11.4.2.2 Node Utilization.*** We measured the node utilization distribution. Figure 11.16 shows that the overall node utilization in the range query test is lower than that in the point query test. This is because the overlapping of query windows makes the distribution of requests among nodes slightly uneven, but we can see only about 10% of nodes are underutilized in our experiment.

### 11.4.3 Growth of the Replicas of an Indexing Node

We visualize the replica creating process for an index node in Figure 11.17 and Figrue 11.18. Figure 11.17 plots how the replica number of an indexing node grows

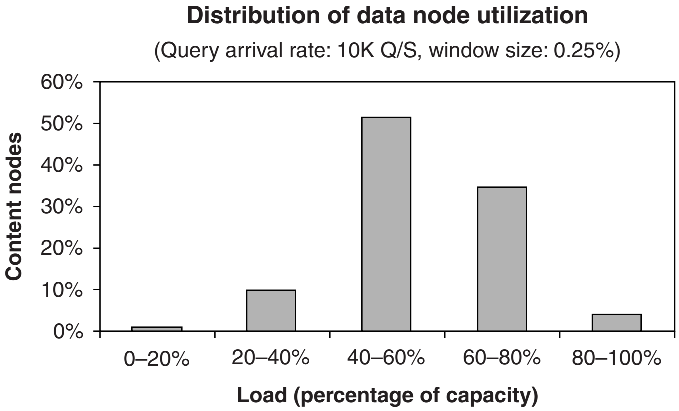


**Figure 11.16** Distribution of data node utilization.

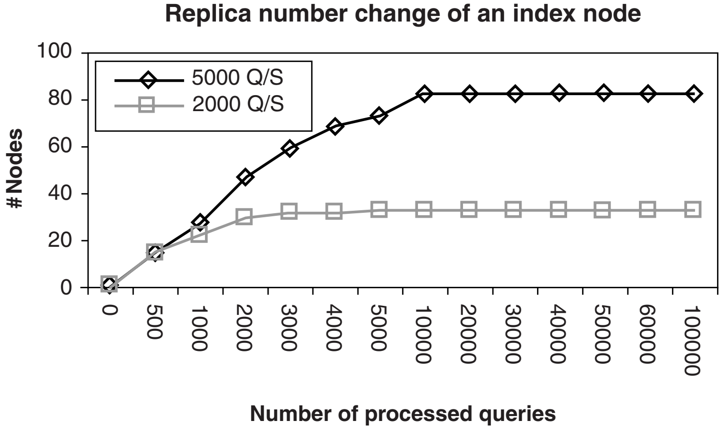


**Figure 11.17** The change of replica number of an index node.

along with the number of queries processed. Figure 11.18 plots how this process affects the query-answering performance. We test this under two different workloads. These figures show that when the replica creation completes and its capacity can serve queries at the given arrival rates, the response time decreases, and the query service quality falls below the predefined threshold (10 queries in the queue in this case).

At the beginning of the test, the queuing time increases rapidly until a replica is created and starts to serve queries. It is important to mention that the replica growth pattern of indexing nodes is not affected by the type of queries (*i.e.*, range or point queries). The replica growth pattern is not affected by query distribution in a point query case either. The experiment shows that our mechanism enables the system to scale up in a self-organizing manner.

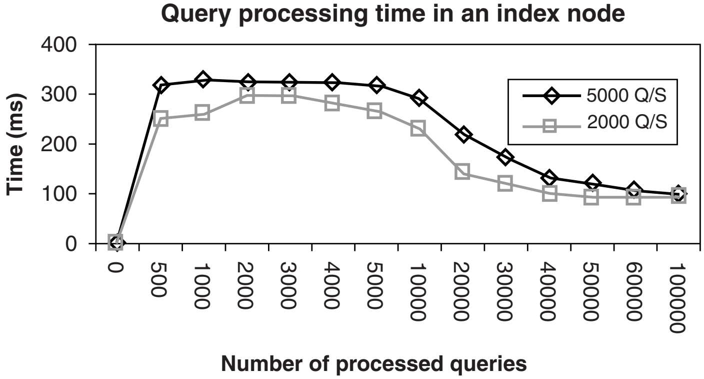


**Figure 11.18** Index query queuing time *vs.* number of processed queries.

## 11.5 RELATED WORK

Our data model is related to work on flexible metadata management in social networking sites. Yu *et al.* [24] proposes a generic data model for tags and data attributes. Differing from their effort on clustering tags and attributes, our focus is scalability of storing and accessing flexibly indexed data.

Multidimensional indexing, including low- and high-dimensional indexing, has received extensive attention in the context of centralized databases [2, 5, 6, 8, 13]. Adapting these methods to the peer-to-peer (P2P) systems encounters four challenging issues: distribution, dynamism, data evolution, and decentralization [10]. There are efforts on partitioning multidimensional data [10, 16] in a P2P environment. As the multidimensional data in our data model contain only location information of the actual data and data indexes are distributed according to index types, an index node in our mechanism does not need to store much information, and therefore, partitioning an R-tree is not of our concern.

Compared with work on supporting range query in a DHT network [1, 4, 12, 15, 17, 18], our approach simply uses R-trees to store range information and maps these trees to DHT nodes corresponding to the attribute set they represent. Our replication mechanism effectively can overcome the single point of failure problem by creating additional replicas at the time an index node is created.

Our mechanism bears some similarities to the Google file system (GFS) [23], which uses a centralized server to store metadata of data chunks distributed in a Google cluster. It differs from GFS in the following aspects:

1. There are many index nodes in our systems, and an index node is only responsible for data indexed using a certain attribute set.
2. The replicas of an index node are managed in a self-organizing manner in our mechanism.

Our data store deals with scalability of data storage and data access simultaneously. There are few approaches to this. In the context of multidimensional indexing schemes, Jagadish *et al.* [9] attempt to address both problems based on migrating partial data from a node. This is inadequate in highly skewed data access distributions. In such cases, a single popular data value can make the node heavily loaded; transferring this value to another peer node only transfers the problem but does not solve. Therefore, the problem should be addressed better using replication of popular values to distribute the access load among the peers storing such replicas.

## 11.6 SUMMARY

In this chapter, we discussed how to manage effectively fast evolving scientific data. We described a flexible data store for managing distributed scientific data. To manage data annotations, which form an important part of bioinformatics data, our data store supports a multidimensional data indexing scheme. The data indexing model is unique because of its flexibility and its support by a underlying self-organized system. The scalability of the data store is addressed through data partitioning and replication. The replication decision is made by each node based on local information, and the load-balancing among replicas is achieved in a decentralized manner. Our extensive simulations showed that the data store scales on demand for both data storing and accessing.

## REFERENCES

1. J. Aspnes and G. Shah. Skip graphs. *Proceeding of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, MD, 2003, pp. 384–393.
2. N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *SIGMOD*, conference, Atlantic City, NJ, 1990, pp. 322–331.
3. J.L. Bentley. Multidimensional binary search trees used for associative searching. *Commun ACM*, 18(9):509–517, 1975.
4. H. Jagadish, B.C. Ooi, and Q.H. Vu. BATON: A balanced tree structure for peer-to-peer networks. *VLDB*, conference, Trondheim, Norway, 2005, pp. 661–672.
5. E. Bertino, B.C. Ooi, R. Sacks-Davis, K. Tan, J. Zobel, B. Shidlovsky, and B. Catania. Indexing Techniques for Advanced Database Applications. Kluwer Academics, New York, 1997.
6. C. Bohm, S. Berchtold, D.A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput Surv*, 33(3):322–373, 2001.
7. A. Mondal, K. Goda, and M. Kitsuregawa. Effective load-balancing via migration and replication in spatial grids. *DEXA*, conference, Prague Czech Republic, 2003, pp. 202–211.
8. A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD*, conference, Boston, MA, 1984, pp. 47–57.
9. H.V. Jagadish, B.C. Ooi, Q.H. Vu, R. Zhang, and A. Zhou. VBI-Tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. *ICDE*, conference, Atlanta, GA, 2006.
10. P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: multidimensional queries in P2P systems. *Proceedings of the Seventh International Workshop on the Web and Databases (WebDB 2004)*, Paris, France, 2004, pp. 19–24.
11. V. Gaede and O. Gunther. Multidimensional access methods. *ACM Comput Surv*, 30(2), 1998, pp. 170–231.
12. O.D. Sahin, A. Gupta, D. Agrawal, and A.E. Abbadi. A peer-to-peer framework for caching range queries. *ICDE*, Boston, MA, 2004, pp. 165–176.
13. T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. *VLDB*, Brighton, England, 1987, pp. 507–518.
14. A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proceedings of Middleware*, Heidelberg, Germany, 2001, pp. 329–350.

15. A. Gupta, D. Agrawal, and A.E. Abbadi. Approximate range selection queries in peer-to-peer systems. *CIDR*, conference, Asilomar, CA, USA, 2003.

16. C. Zhang, A. Krishnamurthy, and R.Y. Wang. Skipindex: towards a scalable peer-to-peer index service for high dimensional data. *Technical Report*, Princeton University, 2004.

17. T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, load balancing and efficient range query processing in DHTs. *EDBT*, Munich, Germany, 2006, pp. 131–148.

18. A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. P-Tree: A P2P index for resource discovery applications. *WWW*, New York, NY, 2004, pp. 390–391.

19. Y.J. Kim and J.M. Patel. Rethinking choices for multi-dimensional point indexing: making the case for the often ignored quadtree. *CIDR*, Asilomar, CA, 2007, pp. 281–291.

20. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SOSP*, Stevenson, WA, 2007, pp. 205–220.

21. F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. *OSDI*, Seattle, WA, 2006, pp. 205–218.

22. C. Wang, B.A. Alqaralleh, B.B. Zhou, F. Brites, and A.Y. Zomaya. Self-organizing content distribution in a data indexed DHT network. *P2P*, Cambridge, England, 2006, pp. 241–248.

23. S. Ghemawat, H. Gobioff, and S-T. Leung. The Google file system. *SOSP*, Bolton Landing, NY, 2003, pp. 29–43.

24. B. Yu, G. Li, B.C. Ooi, and LZ. Zhou. One table stores all: Enabling painless free and easy data publishing and sharing. *CIDR*, Asilomar, CA, 2007, pp. 142–153.

# 12

# ALGORITHMS FOR THE ALIGNMENT OF BIOLOGICAL SEQUENCES

Ahmed Mokaddem and Mourad Elloumi

## 12.1 INTRODUCTION

Bioinformatics is a science dedicated to the automatic processing of information related to biological macromolecules (*i.e.*, DNA, RNA, and proteins). These macromolecules are coded by strings called *biological sequences*. Every character in a string codes a constituent of the macromolecule. DNA, RNA, and proteins can be coded by sequences in which every character is in {A, T, C, G}, {A, U, C, G}, and {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}, respectively. Among the most studied problems in bioinformatics is the *comparison* of biological sequences in order to identify similar substrings, occuring in the same order, in these sequences. This operation makes a very important contribution in the analysis of biological macromolecules. In fact, it can reveal information about shared functions of biological macromolecules, coming from several different organisms, by the identification of regions that are shared by the sequences coding these macromolecules. These regions, which have been conserved during evolution, often play an important structural or functional role and, consequently, shed light on the mechanisms and the biologic processes in which these macromolecules participate. In addition, the comparison of biological sequences permits the detection of functional regions. It is also used in evolutionary studies to analyze relationships that exist between species and establish whether two, or several, macromolecules are *homologous* (*i.e.*, have a

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

common biological ancestor) and to reconstruct the *phylogenetic tree* that links them to this ancestor. Another application of the comparison of biological sequences is the prediction of secondary/tertiary structures by comparing the sequences of biological macromolecules with known secondary/tertiary structures to those with unknown ones. The comparison of biological sequences also facilitates the classification of these sequences in different families according to the shared similar regions. The comparison of biological sequences can be achieved *via aligning* these sequences; it consists in optimizing the number of matches between the characters occurring in the same order in each sequence. We distinguish two main classes of alignments:

1. *Global alignments*: A *global alignment* involves the alignment of entire sequences. Global alignments are suitable when the sequences to compare are *closely* related.
2. *Local alignments*: A *local alignment* involves the alignment of portions of sequences. Local alignments are suitable when the sequences to compare are *distantly* related.

Although optimal algorithms exist for the alignment of two sequences, also called *pairwise alignment*, the problem of aligning more than two sequences, also called *multiple alignment*, is Nondeterministic Polynomial (NP)-complete [14,29].

## 12.2 ALIGNMENT ALGORITHMS

In the next subsection, we present pairwise alignment algorithms.

### 12.2.1 Pairwise Alignment Algorithms

There are two types of pairwise alignment algorithms: *pairwise global alignment* algorithms and *pairwise local alignment* ones. Let us begin with *pairwise global alignment* algorithms.

***12.2.1.1 Pairwise Global Alignment Algorithms.*** There are two main approaches to construct a pairwise global alignment:

The *Dynamic programming approach* [6,7]: The most used dynamic programming algorithm for pairwise global alignment is the one of Needleman and Wunsch [58]. By using this algorithm, the construction of a pairwise global alignment of two sequences $S_1$ and $S_2$, with respective lengths $m$ and $n$, is performed in two steps:

1. During the first step, we construct a matrix $M$ of size $mn$, and we initialize it by using a *substitution matrix* (*e.g.*, *Percent Accepted Mutations* (PAM) [21], *Blocks Substitution Matrix* (BLOSUM) [38]). Then, we transform matrix $M$ by adding scores line by line, starting by the rightmost lower cell and ending

**Table 12.1 Gap penalties, where *k* is the number of successive gaps, and *a*, *b*, and *c* are constants**

| | |
|---|---|
| *Linear gap penalty* | $P = ak$ |
| *Affine gap penalty* | $P = ak + c$ |
| *Logarithmic gap penalty* | $P = b\log(k) + c$ |
| *Logarithmic-affine gap penalty* | $P = ak + b \times \log(k) + c$ |

by the leftmost upper one, by using the following equation:

$$M[i, j] = se(i, j) + \max(M[x, y]) \tag{12.1}$$

where $x = i + 1$ and $j < y \leq n$ or $i < x \leq m$ and $y = j + 1$, and $se(i, j)$ is the score between the character at position $i$ in $S_1$ and the one at position $j$ in $S_2$. We also can incorporate in the equation a *gap* penalty. A *gap* is a character (*e.g.*, $-$), inserted in aligned sequences so that aligned characters are found in front of each other. It is sufficient to subtract from the calculation of every sum a penalty according to their position. So, equation (12.1) becomes:

$$M[i, j] = se(i, j) + \max\begin{pmatrix} M[i+1, j+1], \\ M[x, j+1] - P, \\ M[i+1, y] - P \end{pmatrix} \tag{12.2}$$

where $i + 2 < x \leq m$ and $j + 2 < y \leq n$, and $P$ is a gap penalty.
The gap penalty $P$ can have several possible forms. Examples of gap penalties are given in Table 12.1.

2. During the second step, we establish a path in the matrix, called *maximum scores path*, which leads to an optimal pairwise global alignment. The construction of this path is achieved by starting from the cell that contains the maximum score in the transformed matrix, which corresponds normally to the leftmost upper cell and allows three types of possible movements (see Figure 12.1):

   (i) *Diagonal movement*: This movement corresponds to the passage from a cell $(i, j)$ to a cell $(i + 1, j + 1)$.

   (ii) *Vertical movement*: This movement corresponds to the passage from a cell $(i, j)$ to a cell $(i + 1, j)$.

   (iii) *Horizontal movement*: This movement corresponds to the passage from a cell $(i, j)$ to a cell $(i, j + 1)$.

   The time complexity of the algorithm of Needleman and Wunsch is $O(mn)$. Other dynamic programming algorithms for pairwise global alignment exist, such as the one of Huang and Chao [40] and NGILA [16].

The *Anchoring approach*: Pairwise global alignment algorithms that adopt this approach operate as follows: First, they search for identical or similar regions in the two sequences by using different techniques such as *suffix trees* and *dot*

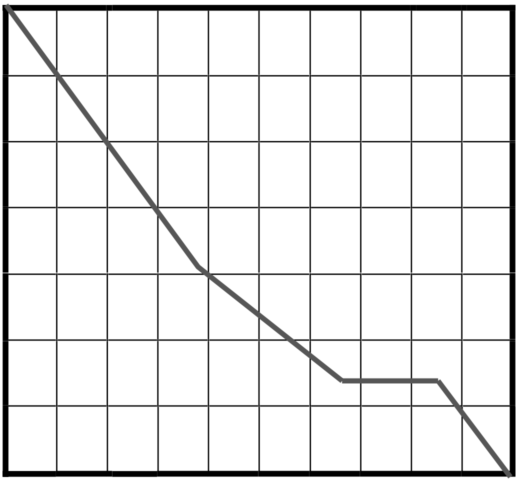


**Figure 12.1** Backtracking in a two-dimensional matrix.

*matrices* or by using a local alignment algorithm such as CHAOS [13]. These regions are called *anchors*. Then, they form the final alignment by chaining the anchors identified in the previous step. Finally, they align the regions situated between the anchors by using a standard dynamic programming algorithm or by applying the same procedure, by recursive calls, or by combining both. Compared with the dynamic programming approach, the anchoring approach is economic in memory space, especially when applied to long sequences.

Among the pairwise global alignment algorithms that adopt the anchoring approach, we can cite MUMMER [22], AVID [10], which uses *suffix trees* to detect anchors, GLASS [5], LAGAN [12], which uses the CHAOS algorithm [13], and ACANA [41].

***12.2.1.2 Pairwise Local Alignment Algorithms.*** There are two main approaches to construct a pairwise local alignment:

The *Dynamic programming approach*: The most used dynamic programming algorithm for pairwise local alignment is the one of Smith and Waterman [77]. The main difference with the algorithm of Needleman and Wunsch [58] is that any cell of the matrix $M$ can be considered as a starting point for the calculation of the scores and that any score that becomes lower than zero stops the progression of the calculation of the scores. The associated cell is then reinitialized to zero and can be considered as a new starting point. That implies that the selected system of scores has negative scores for bad associations that can exist between the characters of the sequences. The equation used for the calculation of each score during the transformation of the initial matrix is as follows:

$$M[i, j] = \max \begin{pmatrix} se(i, j) + M[i + 1, j + 1], \\ se(i, j) + \max(M[x, j + 1] - P), \\ se(i, j) + \max(M[i + 1, y] - P), \\ 0 \end{pmatrix} \tag{12.3}$$

where $i + 2 < x \leq m$ and $j + 2 < y \leq n$, $se(i, j)$ is the score between the character at position $i$ in $S_1$ and the one at position $j$ in $S_2$, $P$ is a gap penalty, and $m$ and $n$ are the lengths of the sequences $S_1$ and $S_2$ to align, respectively.

The time complexity of the algorithm of Smith and Waterman is $O(mn)$.

The *Seeding approach:* Pairwise local alignment algorithms that adopt this approach use a hashing function to define a *seed* and use it as a model to detect alignments. A *seed* is a substring made-up by characters that can be contiguous or not and defined on a precise alphabet. A seed is characterized by its *extent*, which represents the length of the substrings that can be covered by the seed, and by its *weight*, which represents the number of characters that must appear simultaneously in the seed and in the substrings covered by the seed. These characters are called *matches*. A seed can be represented either by a set $\{i, i, i, \ldots\}$, where $i$ is a position of a match, or by a substring defined on alphabets like {#, -},{#, @, -} or {0,1}, where "#" or "1" represents a match, "-" or "0" represents a joker character, and "@" represents the characters associated with the following substitutions ("G" with "C" or "A" with "T").

The seeding approach is based on the notion of *filtering*: It involves, first, the deletion of the zones that have no possibility of participating in the final local alignment and, second, the conservation of the positions that verify the seed.

The principle of the seeding approach first comprises defining the seed used for filtering then, detecting in every sequence the positions of the regions that verify the seed. Among pairwise local alignment algorithms that adopt the seeding approach, we cite FASTA [64], BLAST [1], PATTERNHUNTER [53], CHAOS [13], YASS [59], and BLASTZ [73]. Table 12.2 lists discussed pairwise alignment softwares.

### 12.2.2 Multiple Alignment Algorithms

In the next subsection, we present multiple alignment algorithms. There are two main types of multiple alignment algorithms: *multiple global alignment* algorithms and *multiple local alignment* ones. However, there are also algorithms that combine local and global alignment.

***12.2.2.1 Multiple Global Alignment Algorithms.*** There are four main approaches to construct a multiple global alignment:

The *Dynamic programming approach*: A possible exact dynamic programming algorithm for the multiple global alignment problem is a generalization of the

**Table 12.2 Pairwise alignment softwares**

| Software | Type | Approach | Link |
|---|---|---|---|
| ACANA [41] | Global | Anchoring | http://biomedempire.org |
| AVID [10] | Global | Anchoring | http://baboon.math.berkeley.edu/avid_supplementary/avid.html |
| BLAST [1] | Local | Seeding | http://www.ebi.ac.uk/Tools/blast/ |
| CHAOS [13] | Local | Seeding | http://www.cs.toronto.edu/~brudno/chaos/ |
| FASTA [64] | Local | Seeding | http://www.ebi.ac.uk/Tools/fasta/ |
| LAGAN [12] | Global | Anchoring | http://genome.lbl.gov/cgi-bin/VistaInput?align_pgm=lagan&num_seqs=2 |
| MUMMER [22] | Global | Anchoring | http://mummer.sourceforge.net/ |
| NGILA [16] | Global | Dynamic Programming | http://scit.us/projects/ngila/ |
| YASS [59] | Local | Seeding | http://bioinfo.lifl.fr/yass/yass.php |

pairwise global alignment algorithm of Needleman and Wunsch [58]. Multiple global alignment algorithms that are based on a dynamic programming approach operate in two steps:

1. During the first step, they fill a matrix $M$ of size $L^N$, where $N$ is the number of the sequences and $L$ is the maximal length of a sequence.
2. Then, during the second step, they make a backtracking in the matrix $M$ to construct a maximal score path that corresponds to an optimal alignment (see Figure 12.2).

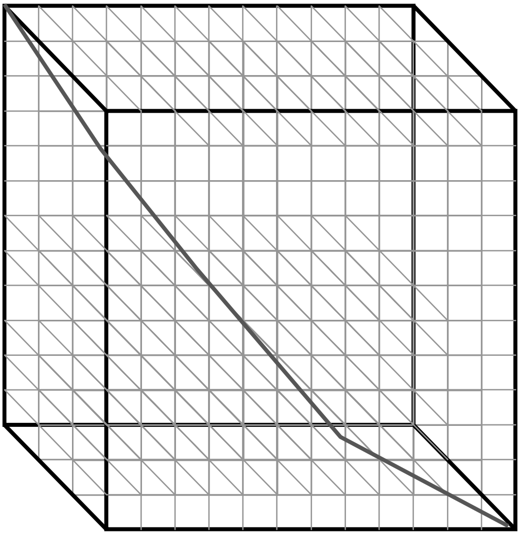


**Figure 12.2** Backtracking in a three-dimensional matrix.

An exact dynamic programming algorithm for the multiple global alignment problem is exponential in computing time and in memory space. A heuristic dynamic programming algorithm has been developed and implemented in the Measurement Systems Analysis (MSA) software [52]. The MSA software can handle only a few short sequences and is not often used in practice.

The *Progressive approach*: By using the progressive approach, we construct a multiple alignment in a gradual manner. First, we construct alignments for each pair of sequences in the set of $N$ sequences to align by using a pairwise alignment algorithm. Then, we assign to each pairwise alignment a score by using a score function. Finally, we construct the final multiple alignment by using an algorithm based on the pairwise alignments obtained. The two most similar sequences are aligned then; at each iteration, we align the aligned sequences with another sequence chosen according to precise criteria. Hence, we gradually align larger and larger sets of sequences until all the $N$ sequences are aligned.

The progressive approach is the most used multiple global alignment approach, and several algorithms have been proposed. These algorithms differ according to the following criteria:

1. The pairwise alignment algorithm (dynamic programming, anchoring)
2. The score function or the distance between a pair of sequences (score [14,61], substitution matrix [21,38], . . . )
3. The order of merging sequences in the final alignment (*guide tree* [78,71], . . . )
4. The method of aligning sequences during the multiple alignment stage (aligning *profiles* [34], aligning alignments [44,45,94], . . . )

Multiple global alignment algorithms adopting the progressive approach are fast, simple to implement, and require a small memory space. However, they present two major drawbacks:

1. The restriction of only comparing two sequences at a time rather than comparing all sequences simultaneously does not enable taking into consideration the common characters to a set of sequences.
2. The constructed alignment depends on the order in which the sequences are aligned and on the chosen score.

Among multiple global alignment algorithms adopting the progressive approach, we cite the one of Feng and Doolitle [31], CLUSTALW [85], T-COFFEE [62], DBCLUSTAL [86], MAVID [11], KALIGN [48], PRALINE [75], SPEM [97], EXPRESSO [3], PSALIGN [83], COBALT [63], PROMALS [67], and GRAMALIGN [70].

To address the drawbacks of the progressive approach, a stage of refinement of the multiple global alignments often is applied. Multiple global alignment algorithms adopting the progressive approach that apply a refinement stage are called *hybrid algorithms*. Different strategies of refinement have been developed (*e.g.*, the iterative construction of the *guide tree* until the stabilization of this tree, or the division of the *guide tree* in subtrees followed by the alignment of the sequences of each subtree, separately, and the integration of the obtained alignments).

Hybrid algorithms include MULTALIN [20], PRRP [36], MULTI-LAGAN [13], MUSCLE [28], PROBCONS [25], MAFFT [43], and MSAID [54].

The *Iterative approach*: Algorithms adopting this approach construct an initial alignment, then during each iteration, they perform a set of modifications on the current alignment to construct a new one. The iterations are repeated until *convergence* (*i.e.*, no improvement can be made on the current alignment). Several ways have been described to modify an alignment [92]:

1. A disruption such as an insertion/deletion of one or more gaps is made in the alignment.
2. One or more sequences are excluded from the initial alignment. The remaining sequences are realigned, and finally, the new alignment is aligned with the excluded sequences.
3. The alignment is divided into two groups, then each group is aligned separately, and finally, the two alignments are aligned.
4. The alignment is divided into two groups, then a profile is constructed for each alignment, and finally, the two profiles are aligned.

The iterative approach is especially suitable for stochastic algorithms such as *genetic algorithms* and *simulated annealing ones*. It can improve the quality of an alignment but needs more computing time than the progressive approach.

Multiple global alignment algorithms adopting the iterative approach also can incorporate *Hidden Markov Models* (HMM). Among these algorithms, we cite HMMER [26], SAM [42], SATCHMO [27], and FSA [9]. The time complexities of SAM and SATCHMO are, respectively, $O(L^2NK)$ and $O(L^2N^2 + LN^3)$, where $N$ is the number of the sequences, $L$ is the length of a sequence, and $K$ is the number of the iterations.

Other multiple global alignment algorithms adopting the iterative approach include SAGA [60] and QOMA [96].

The *Divide-and-conquer approach*: The multiple global alignment algorithms that adopt this approach process the sequences to align simultaneously. By using these algorithms we operate in three steps. First, we choose a position in each sequence, which subdivides the sequence into two smaller ones: a prefix and a suffix. We thus obtain two new families of sequences: the family of prefixes and the family of suffixes. Then, we reiterate recursively this operation on the new families until we obtain small sequences that can be aligned optimally. Finally, the alignment of the initial sequences is obtained by concatenating the alignments of the small ones.

The drawback of the divide-and-conquer approach is that the choice of the division positions in each sequence has an influence on the construction of the alignements of the generated families of prefixes and suffixes.

Multiple global alignment algorithms that adopt the divide-and-conquer approach include DCA [80] and a-$\mu$ALIGN [30]. The time complexity of a-$\mu$ALIGN is $O(N^2L^2(\log(L))^2)$, where $N$ is the number of the sequences and $L$ is the length of the longest sequence.

A new category of multiple global alignment algorithms, called *constrained multiple alignment algorithms*, tries to improve the biological significance of an alignment by integrating structural and/or functional information, extracted from different biological databases, or by using programs of comparison of secondary/tertiary structures. Several techniques have been proposed to improve the biological significance of an alignment, among which, we mention:

1. Alignment of significant motifs extracted from biological databases in each sequence. The remainder of the sequences is aligned by using one of the methods described earlier
2. Construction of a multiple global alignment by using local structural alignment of secondary/tertiary structures

Constrained multiple alignment algorithms include DBCLUSTAL [86], FMALIGN [17], PRALINE [75], SPEM [97], EXPRESSO [3], MUMMALS [66], PSALIGN [83], HSA [95], COBALT [63], and PROMALS [67].

**12.2.2.2 Multiple Local Alignment Algorithms.** There are two main approaches to construct a multiple local alignment:

1. *Dynamic programming approach*: A possible exact dynamic programming algorithm for the multiple local alignment problem is a generalization of the pairwise local alignment algorithm of Smith and Waterman [77]. An exact dynamic programming algorithm for the multiple local alignment problem is exponential in computing time and in memory space.
2. *Motif finding approach*: Multiple local alignment algorithms adopting this approach are based on the search of *motifs* in the set of sequences to align. These algorithms are generally statistical ones. The algorithms adopting the motif finding approach include MACAW [74], MATCHBOX [23], GIBBS [50], MEME [4], and GLAM [32].

Other multiple local alignment algorithms have been defined and are based on other techniques. Among these algorithms, we mention DIALIGN [56], which is based on the use of *dotplots* to extract common fragments in each pair of sequences, TSUKUBA-BB [39], which is based on a *branch-and-bound approach*, POA [51], which is based on a graph representation of an alignment, CHAOS/DIALIGN [13], which is based on the CHAOS algorithm to search for similar regions and the DIALIGN principle of consistency, and *Align-m* [91].

Let us note that there are algorithms that refine multiple alignments to improve the quality of these alignments and to make them more meaningful. Among these algorithms, we mention:

1. RASCAL [88]: The RASCAL algorithm operates in two steps. During the first step, it analyzes the initial multiple alignment by localizing the well-aligned regions by using the Mean Distance (MD) score. Then, during the second step, it detects the badly aligned regions and realigns them.
2. REFINER [18]: The REFINER algorithm realigns each sequence with the profile of the alignment of the remaining sequences. The displacement of blocks in the sequences is allowed under the condition that a block does not contain any gaps. Convergence is obtained when all iterations are applied or when no improvement is observed in the alignment.
3. RF [92]: The RF algorithm is based on an approach similar to the one on which REFINER is based but without any condition on the blocks. The iterations are stopped if convergence is obtained or if the number of the iterations reaches $2N^2$, where $N$ is the number of the sequences.

Table 12.3 lists discussed multiple alignment software.

## 12.3 SCORE FUNCTIONS

A *score function*, or *objective function*, is a function that assigns to an alignment a score that reflects its quality or significance. The alignment that has the maximal score is considered to be an *optimal* alignment. Score functions thus optimize mathematical scores and do not necessarily reflect a biological significance. The most widely used score functions are:

1. *Sum of pairs* [14]: The *Sum of Pairs* (SP) score is used by most alignment softwares. The SP score corresponds to the sum of the scores for all pairs of characters in the alignment and is defined by the following equation:

$$\mathrm{SP}(A) = \sum_{i=1}^{L} \sum_{1<k<j<N} s(w_k[i], w_j[i]) \tag{12.4}$$

   where $w_k[i]$ and $w_j[i]$ are the characters in the sequences $k$ and $j$ that are in the $i$th column of the alignment $A$, $L$ is the length of the alignment $A$, and $s$ is the score for aligning a pair of characters.
2. *Weighted sum of pairs* [35]: The *Weighted Sum of Pairs* (WSP) score is a variant of the SP score that assigns a weight to each sequence according to its importance in the alignment. This weight depends on the relationships

Table 12.3 Multiple alignment softwares

| Software | Type | Approach | Link |
|---|---|---|---|
| CLUSTAL W [85] | Global | Progressive | www.clustal.org/ |
| COBALT [63] | Global | Progressive | ftp://ftp.ncbi.nlm.nih.gov/pub/agarwala/cobalt |
| DBCLUSTAL [86] | Global | Progressive | ftp://ftp-igbmc.u-strasbg.fr/pub/DbClustal |
| DCA [80] | Global | Divide-and-Conquer | http://bibiserv.techfak.uni-bielefeld.de/dca/submission.html |
| DIALIGN [56] | Local | Motif Finding | http://bibiserv.techfak.uni-ielefeld.de/dialign/submission.html |
| EXPRESSO [3] | Global | Progressive | http://www.tcoffee.org |
| FSA [9] | Global | Iterative | http://orangutan.math.berkeley.edu/fsa |
| GLAM [32] | Local | Motif Finding | http://meme.sdsc.edu/meme4/cgi-bin/glam2.cgi |
| Gramalign [70] | Global | Progressive | http://bioinfo.unl.edu/GramAlign.html |
| HMMER [26] | Global | Iterative | http://hmmer.janelia.org |
| KALIGN [48] | Global | Progressive | http://www.ebi.ac.uk/Tools/kalign/ |
| MAFFT [43] | Global | Progressive/Iterative | http://align.bmr.kyushu-u.ac.jp/mafft/online/server |
| MATCHBOX [23] | Local | Motif Finding | http://www.fundp.ac.be/sciences/biologie/bms/matchbox_submit.shtml |
| MAVID [11] | Global | Progressive | http://baboon.math.berkeley.edu/mavid |
| MEME [4] | Local | Motif Finding | http://meme.sdsc.edu/meme4/cgi-bin/meme.cgi |
| MSA [52] | Global | Dynamic Programming | http://xylian.igh.cnrs.fr/msa/msa.html |
| MULTALIN [20] | Global | Progressive/Iterative | http://bioinfo.genotoul.fr/multalin/multalin.html |
| MULTI-LAGAN [12] | Global | Iterative | http://lagan.stanford.edu/lagan_web/index.shtml |
| MUMMALS [66] | Global | Progressive/Iterative | http://prodata.swmed.edu/mummals/mummals.php |
| MUSCLE [28] | Global | Progressive/Iterative | http://www.drive5.com/muscle |
| PLASMA [24] | Global | Progressive | http://genoweb1.irisa.fr/Serveur-GPO/outils_acces.php3?id_syndic=259&lang=en |
| PSALIGN [83] | Global | Progressive | http://faculty.cs.tamu.edu/shsze/psalign |
| PRALINE [75] | Global | Progressive | http://www.ibi.vu.nl/programs/pralinewww |
| PROBCONS [25] | Global | Progressive/Iterative | http://probcons.stanford.edu/index.html |
| PROMALS [67] | Global | Progressive | http://prodata.swmed.edu/promals/promals.php |
| PRRP/PRRN [36] | Global | Progressive/Iterative | http://prrn.hgc.jp |
| SAGA [60] | Global | Iterative | http://www.tcoffee.org/Projects_home_page/saga_home_page.html |
| SATCHMO [27] | Global | Iterative | http://www.drive5.com/lobster/index.htm |
| SPEM [97] | Global | Progressive | http://sparks.informatics.iupui.edu/Softwares-Services_files/spem.htm |
| T-COFFEE [62] | Global | Progressive | http://www.tcoffee.org |

between the sequences. The WSP score of an alignment is given by the following equation:

$$\mathrm{WSP}(A) = \sum_{i=1}^{L} \sum_{1<k<j<N} p_j p_k s(w_k[i], w_j[i]) \tag{12.5}$$

where $p_j$ and $p_k$ are the weights of the sequences $j$ and $k$, respectively.

3. *Entropy:* The *entropy* defines the frequencies of appearance of each character in each column of the multiple alignment. Thus, the *entropy* of a multiple alignment is the sum of the *entropies* of each column and is defined by:

$$E(A) = -\sum_{i} \sum_{x} p_{x,i} \log p_{x,i} \tag{12.6}$$

where $p_{x,i}$ is the number of occurrences of character $x$ in the $i$th column.

4. *Consensus*: The computation of the *consensus score* requires the construction of the *consensus sequence* of the multiple alignment. The *consensus sequence* is made by taking from each column of the multiple alignment the most frequent character in the column. The *consensus score* is the sum of the scores between each sequence $S_i$, $1 \leq i \leq N$, of the multiple alignment and the consensus sequence $S_c$:

$$Consensus(A) = \sum_{i=1}^{N} s(S_i, S_c) \tag{12.7}$$

where $s$ is the distance between $S_i$, $1 \leq i \leq N$, and $S_c$.

Other scores have been proposed such as *Coffee* [61], *Al2co* [65], *Normd* [87], *Divaa* [69], *Mumsa* [49], and *Confind* [76].

## 12.4 BENCHMARKS

To evaluate the performances of an alignment algorithm, *reference alignments* have been constructed in a manual, or automatic, way with the help of biologists and have been grouped to form *benchmarks*. A benchmark generally includes a ratio for comparing alignments, built by an alignment algorithm, with reference alignments found in the benchmark. For example, we have:

1. Column Score (CS) [84] that represents the ratio between the number of correctly aligned columns and the total number of columns in the *core blocks* (*i.e.*, the regions whose alignments are known).
2. Sum-of-Pairs Score (SPS) [84] that represents the ratio between the number of correctly aligned pairs of characters and the total number of pairs of characters in the core blocks.

Other scores to compare multiple alignments include *overlap score* [47], *cline score* and *shift score* [19], and mean opinion score (MOS) [49].

The most used benchmarks include:

1. BALIBASE [89]: This benchmark is the first benchmark dedicated to protein multiple alignment algorithms and contains several accurate reference alignments. The alignments are constructed based on the superposition of proteins tertiary structures and manual improvement of the results. These alignments are grouped in different categories according to the nature of the set of the sequences used. Thus, each reference set represents a different alignment problem. For example, *reference 1* contains small alignments of sequences with different sizes, whereas *reference 2* is made up of families of sequences aligned with one, two, or three *orphan* sequences.
   BALIBASE uses the CS and SPS scores as criteria for assessment.
   No alignment algorithms give the best result for all references of BALIBASE, but there are algorithms such as PROBCONS, MUSCLE, MAFFT, and T-COFFEE that have good performances.
2. PREFAB [28]: This benchmark is made up of 1932 multiple alignments constructed automatically as follows: The tertiary structures of two sequences are aligned using two different superposition methods. A set of 50 homologous sequences then is extracted from databases, and a multiple alignment is constructed for the whole set of sequences.
   A test alignment is evaluated by using the $Q$ score, which is similar to the SP score of BALIBASE.
3. SABMARK [90]: This benchmark contains families of sequences extracted from the SCOP database [57] of protein structures. It is made up by two sets, *Twilight* and *Superfamily.* These sets contain sequences classified according to the SCOP database. The sequences of *Twilight* are 0–25% identical, whereas those of *Superfamily* are 25–50% identical.
   Every pair of sequences in every set is aligned by superposing their tertiary structures. SABMARK compares the multiple alignments by comparing each pair of sequences in the alignment rather than the complete multiple alignment. Pairs of sequences are extracted from the test alignment, and then each of them is compared with the corresponding pair in the reference alignment. To compare alignments, SABMARK uses the average of the scores of several reference alignments. SABMARK uses the *fd* score, that is identical to the SP score [84] of BALIBASE, and the *fm* score (*i.e.*, the *modeler's score* [72]) that represents the ratio between the number of the correctly aligned pairs of characters and the length of the generated alignment.
4. OXBENCH [68]: This benchmark is constructed automatically using known tertiary structures and different alignment methods. So the result can be biased when we compare alignment algorithms based on the same approach used for the construction of the benchmark. OXBENCH is made up by three subsets:
   a. The first one, called *master*, contains 218 alignments of protein domains of 2 to 122 sequences whose tertiary structures are known.

b. The second one, called *full*, contains all sequences whose domains are represented in *master*.

c. Finally, the last one, called *extended*, is made up by the sequences of *master* in addition to other homologous sequences.

Several scores are implemented in OXBENCH for the assessment of the alignments, such as the CS score and the *position-shift* score.

5. IRMBASE and DIRMBASE [82]: These benchmarks use the same scores as BALIBASE and are dedicated to assessing multiple local alignment algorithms. IRMBASE contains alignments of random synthetic sequences classified in three references and constructed by integrating motifs in the random sequences. These motifs are constructed by using ROSE [81]. DIRMBASE simulates DNA sequences, whereas IRMBASE simulates protein ones.
6. HOMSTRAD [79]: Although this database originally was not designed as a benchmark, it often has been used as such. It contains 1032 alignments of protein sequences representing different structures and grouped in homologous families.
7. DNA PREFAB [15]: This benchmark contains alignments of DNA sequences extracted from different databases.
8. BRALIBASE [33]: This benchmark is the first benchmark used to assess RNA alignment algorithms. It contains alignments of families of noncoding RNA sequences, such as 5S rRNA and tRNA. These sequences are extracted from the RFAM database [37].

Several studies have been conducted to compare different multiple alignment algorithms using different benchmarks and different score functions. Among these studies, we mention [84,47,8]. All these studies show that there are no alignment algorithms that are efficient for all alignment problems, and the choice of the most efficient alignment algorithm depends on the nature and the number of the sequences to align. Table 12.4 lists the popular benchmarks and where to find them.

**Table 12.4 Alignment benchmarks**

| Benchmark | Link |
|---|---|
| BALIBASE [89] | http://www-bio3d-igbmc.u-strasbg.fr/balibase/ |
| BRALIBASE [33] | http://projects.binf.ku.dk/pgardner/bralibase/ |
| HOMSTRAD [79] | http://www-cryst.bioc.cam.ac.uk/ homstrad// |
| IRMBASE and DIRMBASE [82] | http://dialign-t.gobics.de/main |
| PREFAB [28]: | http://www.drive5.com/muscle/prefab.htm |
| OXBENCH [68] | http://www.compbio.dundee.ac.uk/Software/Oxbench/oxbench.htm |
| SABMARK [91] | http://bioinformatics.vub.ac.be/databases/content.html |

## 12.5 CONCLUSION

Sequence alignment is an efficient way to compare biological sequences. It involves the identification of similar substrings in these sequences. Alignment algorithms can be classified according to the type of alignment they perform: either *global* or *local* alignment. The alignment of two sequences is called *pairwise alignment*, whereas the alignment of more than two sequences is called *multiple alignment*.

The multiple alignment problem is NP-complete, and several approaches have been developed to deal with this problem in a polynomial time: the *progressive approach*, the *iterative approach*, and the *divide-and-conquer approach*.

Each approach presents advantages and drawbacks, and the choice of an approach and an algorithm depends on the nature of the sequences to align and the goal of the alignment.

Because most alignment algorithms are heuristic, the evaluation of alignment quality is crucial. The quality of a given alignment is judged by using *score functions* that assign a score that reflects alignment accuracy and significance and that allows the user to differentiate between different alignments of the same set of sequences. The accuracy of an alignment algorithm is measured by using benchmarks that contain reference alignments specifically constructed with the help of biological knowledge. The benchmarks also contain tools for comparing the reference alignments and the results obtained by different alignment algorithms.

Although multiple alignment has been the subject of extensive research, none of the existing alignment algorithms is perfect, and the construction of accurate alignments for numerous complex or distantly related sequences remains a problem that requires new work to meet the new requirements and expectations of the biologists.

The implementation of *intelligent* systems, such as the ALEXSYS [2] expert system, which automatically chooses a suitable algorithm of alignment for a given set of sequences, represents a potential solution and a better alternative to the standard algorithms. The integration of supplementary information, (*e.g.*, structural or functional information) to guide the alignment should improve such systems further.

## ACKNOWLEDGMENTS

We are grateful to Dr. Julie Thompson, of the *Institut de Génétique et de Biologie Moléculaire et Cellulaire,* Strasboug France, for her valuable comments on the manuscript.

## REFERENCES

1. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool, *J Mol Biol*, 215:403–410, 1990.
2. M.R. Aniba, S. Siguenza, A. Friedrich, F. Plewniak, O. Poch, A. Marchler-Bauer, and J.D. Thompson. Knowledge-based expert systems and a proof-of-concept case study for

multiple sequence alignment construction and analysis. *Briefings Bioinformatics*, 10:11–23, 2009.

3. F. Armougom, S. Moretti, O. Poirot, S. Audic, P. Dumas, B. Schaeli, V. Keduas, and C. Notredame. Expresso: Automatic incorporation of structural information in multiple sequence alignments using 3D-Coffee. *Nucleic Acids Res*, 34:W604–W608, 2006.
4. T.L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Mach Learn*, 21(1/2):51–80, 1995.
5. S. Batzoglou, L. Pachter, J. Mesirov, B. Berger, and E.S. Lander. Human and mouse gene structure: Comparative analysis and application to exon prediction. *Genome Res*, 10:950–958, 2000.
6. R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
7. R. Bellman and S. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ, 1962.
8. G. Blackshields, I.M. Wallace, M. Larkin, and D.G. Higgins. Analysis and comparison of benchmarks for multiple sequence alignment. In *Silico Biol*, 6:0030, 2006.
9. R.K. Bradley, A. Roberts, M. Smoot, S. Juvekar, J. Do, C. Dewey, I. Holmes, and L. Pachter. Fast statistical alignment. *PLoS Comput Biol*, 5(5):2009.
10. N. Bray, I. Dubchak, and L. Pachter. AVID: A Global Alignment Program. *Genome Res*, 13:97–102, 2003.
11. N. Bray and L. Pachter. MAVID: Constrained ancestral alignment of multiple sequences. *Genome Res*, 14:693–699, 2004.
12. M. Brudno, C. Do, G. Cooper, M. Kim, E. Davydov, E.D. Green, A. Sidow, and S. Batzoglou. LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic DNA, *Genome Res*, 13:721–731, 2003.
13. M. Brudno, M. Chapman, B. Göttgens, S. Batzoglou, and B. Morgenstern. Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4:2003.
14. H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J Appl Math*, 48(5):1073–1082, 1988.
15. H. Carroll, W. Beckstead, T. O'Connor, M. Ebbert, M. Clement, Q. Snell, and D. McClellan. DNA reference alignment benchmarks based on tertiary structure of encoded proteins. *Bioinformatics*, 23(19):2648–2649, 2007.
16. R.A. Cartwright. Ngila: global pairwise alignments with logarithmic and affine gap costs. *Bioinformatics*, 23(11):1427–1428, 2007.
17. S. Chakrabarti, N. Bhardwaj, P.A. Anand, and R. Sowdhamini. Improvement of alignment accuracy utilizing sequentially conserved motifs. *BMC Bioinformatics*, 5:167–188, 2004.
18. S. Chakrabarti, C.J. Lanczycki, A.R. Panchenko, T.M. Przytycka, P.A. Thiessen, and S.H. Bryant. Refining multiple sequence alignments with conserved core regions. *Nucleic Acids Res*, 34:2598–2606, 2006.
19. M.S. Cline, R. Hughey, and K. Karplus. Predicting reliable regions in protein sequence alignments. *Bioinformatics*, 18(2):306–314, 2002.
20. F. Corpet. Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Res*, 16(22):10881–10890, 1988.
21. M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*. National Biomedical Research Foundation, Washington, DC: 345–358, 1978.

22. A.L. Delcher, A. Phillippy, J. Carlton, and S.L. Salzberg. Fast algorithms for large-scale Genome alignment and comparison. *Nucleic Acids Res*, 30(11):2478–2483, 2002.
23. E. Depiereux and E. Feytmans. *MATCHBOX:* A fundamentally new algorithm for the simultaneous alignment of several protein sequences. *CABIOS*, 8(5):501–509, 1992.
24. V. Derrien, J.M. Richer, and J.K. Hao. PLasMA: un nouvel algorithme progressif pour l'alignement multiple des séquences, Proc. Premières Journées Francophones de Programmation par Contraintes (JFPC'05): 39–48, 2005.
25. C.B. Do, M.S. Mahabhashyam, M. Brudno, and S. Batzoglou. PROBCONS: Probabilistic consistency-based multiple sequence alignment. *Genome Res*, 15:330–340, 2005.
26. S.R. Eddy. Multiple alignment using hidden markov models. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, Cambridge, UK: 114–120, 1995.
27. R.C. Edgar and K. Sjolander. SATCHMO: sequence alignment and tree construction using hidden Markov models. *Bioinformatics*, 19(11):1404–1411, 2003.
28. R.C. Edgar. MUSCLE: multiple sequence alignment with high accuracy high throughput. *Nucleic Acids Res*, 32(5):1792–1797, 2004.
29. I. Elias. Settling the intractability of multiple Alignment. *J Computat Biol*, 13(7):1323–1339, 2006.
30. M. Elloumi and A. Mokaddem. An Algorithm for Multiple and Global Alignments. *Proceedings of the $2^{nd}$ International Conference on Bioinformatics Research and Development, BIRD'08 Vienna, Austria, Communications in Computer and Information Science (CCIS)*, Springer-Verlag, Berlin, Heidelberg, Germany: 479–488, 2008.
31. D.F. Feng and R.F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Biol*, 25(4):351–360, 1987.
32. M.C. Frith, U. Hansen, J.L. Spouge, and Z. Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res*, 32(1):189–200, 2004.
33. P.P. Gardner, A. Wilm, and S. Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res*, 33:2433–2439, 2005.
34. O. Gotoh. Further improvement in methods of group-to-group sequence alignment with generalized profile operations. *CABIOS*, 10:379–387, 1994.
35. O. Gotoh. A weighting system and algorithm for aligning many phylogenetically related sequences. *Comput Appl Biosci*, 11:543–551, 1995.
36. O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J Mol Biol*, 264(4):823–838, 1996.
37. S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S.R. Eddy. RFAM: an RNA family database. *Nucleic Acids Res*, 31(1):439–441, 2003.
38. S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89(22):10915–10919, 1992.
39. P. Horton. TSUKUBA BB: a branch and bound algorithm for local multiple sequence alignment. *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching, CPM'00, (Montréal, Canada), Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, Berlin, Heidelberg, Germany: 84–98, 2000.
40. X. Huang and K.M. Chao. A generalized global alignment algorithm. *Bioinformatics*, 19(2):228–233, 2003.

41. W. Huang, D.M. Umbach, and L. Li. Accurate anchoring alignment of divergent sequences. *Bioinformatics*, 22(1):29–34, 2006.

42. K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies. *Bioinformatics*, 14(10):846–856, 1998.

43. K. Katoh, K. Kuma, H. Toh, and T. Miyata. MAFFT version 5: Improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res*, 33(2):511–518, 2005.

44. J. Kececioglu and W. Zhang. Aligning alignments. *Proceedings of the 9th Symposium on Combinatorial Pattern Matching, CPM'98 (New Jersey, USA), Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, Berlin, Heidelberg, Germany: 189–208, 1998.

45. J. Kececioglu and D. Starrett. Aligning alignments exactly. *Proceedings of the 8th ACM Conference on Research in Computational Molecular Biology, RECOMB'04*, San Diego, CA: 85–96, 2004.

46. T. Lassman, O. Frings, and L.L. Sonnhammer. KALIGN2: high-performance multiple alignment of protein and nucleotide sequences allowing external features. *Nucleic Acids Res*, 37(3):858–865, 2009.

47. T. Lassman and L.L. Sonnhammer. Quality assessment of multiple alignment programs. *FEBS Lett*, 529:126–130, 2002.

48. T. Lassman and L.L. Sonnhammer. KALIGN: An accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics*, 6:298, 2005.

49. T. Lassman and L.L. Sonnhammer. Automatic assessment of alignment quality. *Nucleic Acids Res*, 33:7120–7128, 2005.

50. C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A. Neuwald, and F.J.C. Wootton. Detecting subtle sequence signals : A GIBBS sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

51. C. Lee, C. Grasso, and M.F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

52. D.J. Lipman, S.F. Altschul, and J.D. Kececioglu. A tool for multiple sequence alignment. *Proc Natl Acad Sci U S A*, 86:4412–4415, 1989.

53. B. Ma, T. John, and M. Li. PATTERNHUNTER: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

54. Z. Min, F. Weiwu, Z. Junhua, and C. Zhongxian. MSAID: multiple sequence alignment based on a measure of information discrepancy. *Comput Biol Chem*, 29:175–181, 2005.

55. K. Mizuguchi, C.M. Deane, T.L. Blundell, and J.P. Overington. HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Sci*, 7:2469–2471, 1998.

56. B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, 1999.

57. A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol*, 247:536–540, 1995.

58. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Bio*, 48(1):443–453, 1970.

59. L. Noé and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search, *Nucleic Acids Res*, 33(2):540–543, 2005.

60. C. Notredame and D. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Res*, 24:1515–1524, 1996.

61. C. Notredame, L. Holm, and D.G. Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.
62. C. Notredame, D. Higgins and J. Heringa. T-COFFEE: A novel method for multiple sequence alignments. *J Mol Biol*, 302:205–217, 2000.
63. J.S. Papadopoulos and R. Agarwala. COBALT: constraint-based alignment tool for multiple protein sequences. *Bioinformatics*, 23(9):1073–1079, 2007.
64. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85:2444–2448, 1988.
65. J. Pei and N.V. Grishin. Al2co: Calculation of positional conservation in a protein sequence alignment. *Bioinformatics*, 17(8):700–712, 2001.
66. J. Pei and N.V. Grishin. MUMMALS: Multiple sequence alignment improved by using hidden Markov models with local structural information. *Nucleic Acids Res*, 34(16):4364–4374, 2006.
67. J. Pei and N.V. Grishin. PROMALS: Towards accurate multiple sequence alignments of distantly related proteins. *Bioinformatics*, 23(7):802–808, 2007.
68. G.P. Raghava, S.M. Searle, P.C. Audley, J.D. Barber, and G.J. Barton. OXBENCH: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, 4:2003.
69. D.J. Rodi, S. Mandava, and L. Makowski. Divaa: analysis of amino acid diversity in multiple aligned protein sequences. *Bioinformatics*, 20(18):3481–3489, 2004.
70. D.J. Russell, H.H. Out, and K. Sayood. Grammar-based distance in progressive multiple sequence alignment. *BMC Bioinformatics*, 9:2008.
71. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4:406–425, 1987.
72. J.M. Sauder, J.W. Arthur, and R.L. Dunbrack. Large-scale comparison of protein sequence alignments with structural alignments. *Proteins*, 40:6–22, 2000.
73. S. Schwartz, J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Res*, 13:103–107, 2003.
74. G.D. Schuler, S.F. Altschul, and D.J. Lipman. A workbench for multiple alignment construction and analysis. *Proteins*, 9:180–190, 1991.
75. V.A. Simossis and J. Heringa. PRALINE: A multiple sequence alignment toolbox that integrates homology-extended and secondary structure information. *Nucleic Acids Res*, 33(2):289–294, 2005.
76. J.A. Smagala, E.D. Dawson, M. Mehlmann, M.B. Townsend, R.D. Kuchta, and K.L. Rowlen. Confind: a robust tool for conserved sequence identification. *Bioinformatics*, 21(24):4420–4422, 2005.
77. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J Molecular Biol*, 147:195–197, 1981.
78. P. Sneath and R. Sokal. *Numerical Taxonomy*, Freeman, San Francisco, CA: 230–234, 1973.
79. L.A. Stebbings and K. Mizuguchi. HOMSTRAD: recent developments of the Homologous Protein Structure Alignment Database. *Nucleic Acids Res*, 32, D203–D207, 2004.
80. J. Stoye. Multiple sequence alignment with the divide-and-conquer method. *Gene*, 211:GC45–GC56, 1998.

81. J. Stoye, D. Evers, and F. Meyer. ROSE: generating sequence families. *Bioinformatics*, 14(2):157–163, 1998.

82. A.R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DIALIGN-T: an improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6: 2005.

83. S.H. Sze, Y. Lu, and Q. Yang. A polynomial time solvable formulation of multiple sequence alignment. *J Comput Bio*, 13:309–319, 2006.

84. J.D. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res*, 27(13):2682–2690, 1999.

85. J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680, 1994.

86. J.D. Thompson, F. Plewniak, J.C. Thierry, and O. Poch. DBCLUSTAL: rapid and reliable global multiple alignments of protein sequences detected by database searches. *Nucleic Acids Res*, 28(15):2919–2926, 2000.

87. J.D. Thompson, F. Plewniak, R. Ripp, J.C. Thierry, and O. Poch. Towards a reliable objective function for multiple sequence alignments. *J Mol Biol*, 314:937–951, 2001.

88. J.D. Thompson, J.C. Thierry, and O. Poch. RASCAL: Rapid scanning and correction of multiple sequence alignments. *Bioinformatics*, 19(9):1155–1161, 2003.

89. J.D. Thompson, P. Koehl, R. Ripp, and O. Poch. BALIBASE 3.0: latest developments of the multiple sequence alignment benchmarks. *Proteins*, 61:127–136, 2005.

90. I. Van Walle, I. Lasters, and L. Wyns. SABMARK: a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21(7):1267–1268, 2005.

91. I. Van Walle, I. Lasters, and L .Wyns. Align-m: a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics*, 20(9):1428–1435, 2004.

92. I.M. Wallace, O. O'Sullivan, and D.G. Higgins. Evaluation of iterative alignment algorithms for multiple alignments. *Bioinformatics*, 21(8):1408–1414, 2005.

93. L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J Comput Biol*, 1:337–348, 1994.

94. T.J. Wheeler and J.D. Kececioglu. Multiple alignment by aligning alignments. *Bioinformatics*, 23(13):559–568, 2007.

95. X. Zhang and T. Kahveci. A New Approach for alignment of Multiple Proteins. *Pac Symp Biocomput*: 339–350, 2006.

96. X. Zhang and T. Kahveci. QOMA: quasi-optimal multiple alignment of protein sequences. *Bioinformatics*, 23(2):162–168, 2007.

97. H. Zhou and Y. Zhou. SPEM, Improving multiple sequence alignment with sequence profiles and predicted secondary structure. *Bioinformatics*, 21(18):3615–3621, 2005.

# 13

# ALGORITHMS FOR LOCAL STRUCTURAL ALIGNMENT AND STRUCTURAL MOTIF IDENTIFICATION

Sanguthevar Rajasekaran, Vamsi Kundeti, and Martin Schiller

## 13.1 INTRODUCTION

A protein is characterized by both the amino-acid sequence and the three-dimensional (3-D) structure of the underlying atoms. Although it is a common practice of the biologists to use sequence similarity among different proteins to identify any conserved regions during the evolution, it has been proven that the 3-D structures of the proteins are conserved more fundamentally than the sequence during the evolution. Even though two given proteins may not exhibit much of a sequence homology, the structural similarity between them might account for similar properties. Proteins with a similar structure might have similar properties [10]. This is the motivation behind the study of the structural alignment problem in a manner similar to that of the sequence alignment problem [4].

The structural alignment problem has received immense attention in the past few decades, especially with the increasing number of tertiary structures available in the Protein Data Bank (PDB) [1]. Given two proteins $P_1$ and $P_2$, the problem of structural alignment is to find a highly similar substructure $S_{\text{sub}}$ between $P_1$ and $P_2$. The number of known protein structures has increased drastically from 10,000 in 1999 to 45,000 in 2007. This growth makes manual structural alignment almost impossible,

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

and hence, we need algorithms that can yield almost similar accuracy as manual alignment and are very fast.

Almost all existing algorithms perform structural alignment based on the backbone of the protein. For any two given proteins, these algorithms try to find the correspondence between the $C\alpha$ atoms on the backbone along with the transformation matrices $R$ (rotational) and $T$ (translational) that will transform one protein to the other minimizing the interatomic distance between the corresponding $C\alpha$ atoms (see e.g., [3], [7], [5], [8], and [6]). All these algorithms share a common flavor that consists of two major steps. The first step consists of identifying small structurally similar regions between the two protein back bones. These are known as *alignment fragment pairs* (AFPs). In the next step, a subset of these AFPs is identified that essentially forms the alignment between the two structures. In all these algorithms, the AFPs are identified by sliding a window of *constant* size along the backbone of the protein. However, we feel that using a constant size window in identifying the AFPs is too restrictive, especially when we want to improve the accuracy of structural classification. In this work, we add an extra degree of freedom in the form of *variable length alignment fragment pairs* (VLAFPs) and present a generalized algorithmic framework for structural alignment. Our framework is independent of the scoring schemes used to score the AFP's. Another important fact is that all existing algorithms only consider the global structural alignment between the two proteins $P_1$ and $P_2$ rather than the local alignment. Local structural alignment can be very effective in the identification of structural motifs.

Our contributions are three fold. First, we introduce a new idea of using VLAFPs in structural alignment; second, we provide new scoring schemes based on *center of gravity* (CG) to identify structurally similar AFPs; and finally, we address the problem of identifying *structural motifs* with our algorithm.

The organization of the chapter is as follows. In Section 13.2, we define the local structural alignment problem. In Section 13.3 we introduce our VLAFP framework. In Section 13.4 we show how we can use center-of-gravity-based scores to identify highly structurally similar AFPs. Section 13.5 describes how the VLAFP framework can aid in the identification of structural motifs. Section 13.6 describes how we can classify the proteins based on the VLAFP framework and the center of gravity scoring scheme.

## 13.2 PROBLEM DEFINITION OF LOCAL STRUCTURAL ALIGNMENT

**Input:** Input are two protein structures $P_1 = (a_{1,1}, a_{1,2}, a_{1,3}, \ldots)$ and $P_2 = (b_{1,1}, b_{1,2}, b_{1,3}, \ldots)$, where $a_{i,j}$ represents the $j$-th atom of the $i$-th residue of $P_1$, and $b_{p,q}$ represents the $q$-th atom in the $p$-th residue of $P_2$. In fact, $a_{i,j}$ and $b_{p,q}$ have information about the location of the corresponding atoms. For instance, $a_{i,j} = (X_j, Y_j, Z_j)$ and $b_{p,q} = (X_q, Y_q, Z_q)$.

**Output:** Define the correspondence between $P_1$ and $P_2$ as $C_{1,2} = ((a_{p,q}, b_{r,s}), (a_{m,n}, b_{k,l}), \ldots)$ (*i.e.*, specify which atom of $P_1$ corresponds to which atom of $P_2$). The local structural alignment problem is to find a correspondence ($C_{1,2}$) between $P_1$

and $P_2$ along with a rotation matrix $R$ and a translation matrix $T$ such that when we apply $R$ and $T$ to one set of coordinates $(a_{p,q}, a_{m,n}, \ldots)$, we end up with the other set $(b_{r,s}, b_{k,l}, \ldots)$. The optimization version of this problem is to find a correspondence $C_{1,2}$ such that $|C_{1,2}|$ is maximal.

## 13.3 VARIABLE-LENGTH ALIGNMENT FRAGMENT PAIR (VLAFP) ALGORITHM

The existing algorithms for structural alignment share a common flavor that consists of two major steps. The first step consists of identifying small structurally similar regions between the two protein back bones (AFPs). In the next step, a subset of these AFPs is identified. This subset forms the alignment between the two structures. The following sections will give a brief overview of these steps and the details about our *Variable-Length Alignment Fragment Pair* Algorithm.

### 13.3.1 Alignment Fragment Pairs

AFPs in the first step of the existing algorithms are identified by sliding a window $W$ of *constant* size along the protein backbones. Let $B_1 = c^1_{\alpha 1} c^1_{\alpha 2} \ldots c^1_{\alpha n}$ be the backbone of protein $P_1$, and similarly, let $B_2 = c^2_{\alpha 1} c^2_{\alpha 2} \ldots c^2_{\alpha m}$ be the backbone of protein $P_2$. The backbones $B_1$ and $B_2$ now are transformed into two sequences $W_1 = w^1_1 w^1_2 \ldots w^1_{n-k+1}$ and $W_2 = w^2_1 w^2_2 \ldots w^2_{m-k+1}$, where $k$ is the size of the window $W$ and $w^1_i = c^1_{\alpha i} c^1_{\alpha(i+1)} \ldots c^1_{\alpha(i+k-1)}$, $w^2_j = c^2_{\alpha j} c^2_{\alpha(j+1)} \ldots c^2_{\alpha(j+k-1)}$. The *Alignment Fragment Pairs* are defined as $\text{AFP}_{(i,j)} = (w^1_i, w^2_j)$, and each of these AFPs is associated with a normalized cost function $\text{COST}_{(i,j)} \in [0, 1]$. If $\text{COST}_{(p,q)} \leq \epsilon$ (for some appropriate threshold value $\epsilon$), then it indicates that the structure of the $c - \alpha$ atoms in windows $w^1_p$ and $w^2_q$ have very similar structures. In contrast, if $\text{COST}_{(p,q)} > \epsilon$, then the AFP at $(p, q)$ is not structurally similar. A careful analysis of the algorithms CE [8], DALI [7], TM-Align [17], and PSIST [3] reveals that these algorithms only differ in the cost functions associated with the AFPs. For example, DALI and CE use a pairwise $c - \alpha$ distance matrix to compute $\text{COST}_{(i,j)}$. CE also combines some extra statistical information into the cost function. PSIST uses the bond angle information among the $c - \alpha$ atoms within each window. TM-Align uses the TM-score [16] as its cost function.

All existing algorithms work with *constant* size AFPs. Using constant size AFPs is too restrictive in the identification of good *local alignments* among the back bones of the proteins, especially in the presence of noise in estimating the coordinates of $c - \alpha$ atoms during X-ray crystallography. For example, consider an AFP at $(i, j)$ of constant size $k$. Let $\text{COST}_{(p,q)} > \epsilon$. The cost of the same AFP at $(i, j)$ with a different size $k + \gamma$ may be under the threshold of $\epsilon$. Another good example for the need of VLAFPs is the presence of variable length secondary structure elements that consists of *helices* and *sheets*. An important fact to note is that these secondary structure elements are not always of the same size (in terms of the residues). It is possible that a *helix* structure may consist of eight residues in one structure and may consist of 12 residues in another structure. Therefore, the use of constant size AFPs may not yield a good alignment. For example, if we assume that the size of any AFP is fixed to be eight then a *helix* structure of 12 residues could be matched only partially either at the start of the helix or at the fourth position, thus making the local alignment only partial. However, if we allow the AFPs to take variable length such that $2 \leq |W| \leq 8$, then we clearly can produce an alignment of size $4 + 8$ and could match the 12 residue helix exactly. To address such drawbacks with constant size AFPs, we present a much general idea of VLAFPs. The extra degree of freedom is added in the form of an extra variable into the VLAFP cost function which is defined as follows.

$$\text{VCOST}(i, j, q) = \begin{cases} \text{Cost of aligning a fragment of size ``q''} \\ \text{at position ``i'' in } P_1 \text{ and at} \\ \text{position ``j'' in } P_2 \end{cases}$$

$$\begin{array}{ll} \text{VCOST}(i, j, q) \in [0, 1] & \text{Normalized VLAFP Cost} \\ k_1 \leq q \leq k_2 & \text{Range of the VLAFP variable ``q''} \end{array}$$

Our core noniterative dynamic programming framework is independent of any VCOST function. In the later sections, we introduce a new VCOST function based on center of gravity.

### 13.3.2 Finding the Optimal Local Alignments Based on the VLAFP Cost Function

With the definition of the VLAFP cost function in the previous section, we now describe our dynamic programming framework for finding the local structural alignments among the structures. The aim of this dynamic programming formulation is to find the *longest contiguous sequence of VLAFPs* such that the cost of each VLAFP is under the threshold $\epsilon$. Details of the dynamic programming formulation follow. We define the dynamic programming subproblem in the form of VLCS. Variables $i$ and $j$ refer to the indices of the residues in the protein backbones of corresponding proteins.

$$\text{VLCS}(i, j) = \begin{cases} \text{Longest contiguous sequence of VLAFPs} \\ \text{in the backbones of } P_1 \text{ and } P_2 \text{ ending at} \\ \text{the } i\text{-th and } j\text{-th residues, respectively} \end{cases}$$

In our algorithm, we need a two-step initialization. Because the minimum length of the VLAFP is $k_1$, pairs of the kind $(i, j)$ with $i < k_1$ and $j < k_1$ are not of interest.

$$\begin{array}{ll} \text{VLCS}(i, j) & = 0 \\ 1 \leq (i, j) \leq k_1 - 1 & \end{array}$$

In the second initialization step, we consider the first $k_1$ residues from protein $P_1$ and check whether we can align these residues to any part of the protein $P_2$ based on the cost function VCOST as follows. This initialization is similar to the standard sequence alignment initialization.

$$\begin{aligned}&\text{VLCS}(k_1, j) = \begin{cases} k_1 & \textbf{IF } \text{VCOST}(k_1, j) \le \epsilon \\ 0 & \textbf{ELSE} \end{cases}\\ &1 \le j \le |P_2|\end{aligned}$$

The core dynamic programming computation is based on the following equations:

$$\begin{aligned}&\text{QLCS}(i, j, q) = \begin{cases} q + \text{VLCS}(i - q, j - q) \\ \quad \textbf{IF } \text{VCOST}(i, j, q) \le \epsilon \\ \\ 0 \\ \textbf{ELSE} \end{cases}\\ &k_1 + 1 \le i \le |P_1|,\\ &k_1 + 1 \le j \le |P_2|\\ &\text{VLCS}(i, j) = \begin{cases} \max\{\text{QLCS}(i, j, q)\} \\ k_1 \le q \le k_2 \end{cases}\\ &k_1 + 1 \le i \le |P_1|,\\ &k_1 \le j \le |P_2|\\ &\text{Final answer required} = \begin{cases} \max\{\text{VLCS}(i, j)\} \\ 1 \le i \le |P_1| \\ 1 \le j \le |P_2| \end{cases}\end{aligned}$$

After the end of the computation, we end up with the length of the longest contiguous sequence of VLAFPs such that the cost of each VLAFP is within a threshold $\epsilon$. Along with this, we also can compute the exact position in $P_1$ and $P_2$ where this sequence starts. So our VLAFP framework has the following two major steps to compute the local structural alignment:

- Compute the VCOST$(i, j, q)$ function on the backbones of the proteins.
- Compute local structural alignment, which is equivalent to finding a contiguous sequence of VLAFPs in $P_1$ and $P_2$ such that the cost of each VLAFP is under a threshold $\epsilon$.

A pseudocode of the core dynamic programming frame work is illustrated in Algorithm 13.1. Clearly, VCOST$(i, j, q)$ should be such that it takes a value close to 0 for highly structurally similar AFPs and a value close to 1 for structurally dissimilar AFPs. In the next sections, we introduce a new VCOST function based on the center of gravity that has these desired properties.

**Algorithm 13.1** Core VLAFP Algorithm to compute local structural alignments

```
INPUT  : VCOST,$|P_1|$,$|P_2|$
OUTPUT: Length of optimal local alignment and its location
Initialize VLCS
MaxLen = 0
for $i = k_1$ to $|P_1|$ do
    for $j = k_1$ to $|P_2|$ do
        CurrentMax = 0
        for $q = k_1$ to $k_2$ do
            if VCOST$(i, j, q) \leq \epsilon$ then
                if VLCS$(i - q, j - q) + q >$ CurrentMax then
                    CurrentMax = VLCS$(i - q, j - q) + q$
                end
            end
        end
        VCLS$(i, j)$ = CurrentMax
        if CurrentMax > $MaxLen$ then
            MaxLen = CurrentMax
            StartPosition1 = $i$ − CurrentMax + 1
            StartPosition2 = $j$ − CurrentMax + 1
        end
    end
end
return (MaxLen,StartPosition1,StartPosition2)
```

## 13.4 STRUCTURAL ALIGNMENT BASED ON CENTER OF GRAVITY: SACG

One of the ideas that we propose in this work is that of using the sorted distances from the center of gravity to identify AFPs. One of the advantages of using the center of gravity is that we can perform structural alignment not only at the $c - \alpha$ level but also including the side chains. Our main goal is to use the algorithms in this section to identify highly structurally similar AFPs and build a VCOST function and then apply the VLAFP algorithm to compute the local structural alignment. Before presenting the details, we provide a summary of how exactly the structure of any protein is described in the PDB file format [1].

### 13.4.1 Description of Protein Structure in PDB Format

The PDB file for a protein structure is a text description of the 3-D-coordinates of the atoms/residues in the protein. The file consists of a linear list $L_{\text{pdb}}$ of atoms that are a part of the protein and the corresponding 3-D coordinates of each atom. $L_{\text{pdb}} = (a_{1,1}, a_{1,2}, a_{1,3}, \ldots, a_{i,j} \ldots)$, where $a_{i,j}$ is the $j$-th atom in residue $i$. It is noteworthy that the list $L_{\text{pdb}}$ is partially ordered with respect to the residue numbers (*i.e.*, $a_{p,q} < a_{m,n} \iff (p < m)$). Although there is an ordering among the residues, the atoms within a reside may not follow any order. If $L^1{}_{\text{pdb}}$ and $L^2{}_{\text{pdb}}$ are two PDB structure instances of the same protein, then the ordering of the atoms within each residue may be different (though the residues themselves will be in the same order). As an example the atoms in the first residue of $L^1{}_{\text{pdb}}$ may be ordered as $(a_{1,2}, a_{1,1}, a_{1,5}, a_{1,4}, a_{1,3}, \ldots)$, but the atoms in the same residue of $L^2{}_{\text{pdb}}$ may be ordered as $(a_{1,5}, a_{1,1}, a_{1,2}, a_{1,4}, a_{1,3}, \ldots)$. This variation is mainly because of different frames of reference during X-ray crystallography. The variation of the ordering of the atoms within the same residue makes structural alignment algorithms that consider the sidechain conformations nontrivial. In the next sections, we will see how our algorithms overcome this ordering issue when side-chain conformations are considered.

### 13.4.2 Related Work

The problem of checking whether two point sets (in two-dimensional [2D] or 3-D) are rigidly transformable from one to the other is a well-studied problem in computational geometry. This problem is known as *geometric congruence*. Several algorithms for exact geometric congruence were given in [11], [12], [13], and [15]. All these algorithms solve the exact geometric congruence in $O(n \log n)$ time. There is also a more general version of the geometric congruence known as the $\epsilon$-congruence. In this version, we are required to determine whether two given point sets of the same cardinality are rigidly transformable from one to the other within a tolerance of $\epsilon$. The $\epsilon$-congruence problem can be solved in time $O(n^8)$ deterministically (see, *e.g.*, [13]). The problem of $\epsilon$-congruence is related closely to the substructure identification problem, but a run time of $O(n^8)$ may not be practical. In the literature of structural alignment of proteins, several *iterative* dynamic programming-based algorithms have been proposed (see, *e.g.*, [6] and [14]). However, there are several issues on the convergence of these algorithms. In these algorithms, the correspondence between the atoms is changed in every iteration, and hence, it is possible for these algorithms never to converge to an optimal solution. In our algorithm, we first find the substructures that are highly similar, and we will not change this correspondence throughout the algorithm and finally use the VLAFP framework in Section 13.3 to find the longest common substructure among the protein structures.

### 13.4.3 Center-of-Gravity-Based Algorithm

If $P_1$ and $P_2$ are two given proteins with $n$ residues each, a simple algorithm to find the correspondence between $P_1$ and $P_2$ will take $O(n!)$ time. The key idea behind our structural alignment algorithm based on center of gravity is based on the following theorem.

**Theorem 13.1** *Given two 3-D pointsets $S_1$ and $S_2$ each of size n, with $S_1 = \{(x^1{}_1, y^1{}_1, z^1{}_1),\ (x^1{}_2, y^1{}_2, z^1{}_2), \ldots\}$ and $S_2 = \{(x^2{}_1, y^2{}_1, z^2{}_1), (x^2{}_2, y^2{}_2, z^2{}_2), \ldots\}$, we can check whether $S_1$ is a rigid transformation of $S_2$ in $O(n \log n)$ time and $O(n)$ space.*

This directly follows from the Atkinson's algorithm (exact geometric congruence) (see [11]). The proof is based on a very simple fact that the relative position (from any of the points in the point set) of the center of gravity of a set of 3-D points remains unchanged when these 3-D points are transformed by any rigid transformation. The CG for a 3-D point set is defined as follows:

$$S_1 = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots\};$$
$$X_{\text{CG}} = \frac{\sum_{i=1}^{n} x_i}{n}; Y_{\text{CG}} = \frac{\sum_{i=1}^{n} y_i}{n}; Z_{\text{CG}} = \frac{\sum_{i=1}^{n} z_i}{n}$$

If the relative position of the CG with respect to any of the points in the point set changes because of a transformation, then the transformation is not rigid. We use this fact and compute the Euclidean distance of each point from $(X_{\text{CG}}, Y_{\text{CG}}, Z_{\text{CG}})$. Let this distance for the $i$-th point be ${d_i}^{\text{cg}}$.

$${d_i}^{cg} = \sqrt{(X_{\text{CG}} - x_i)^2 + (Y_{\text{CG}} - y_i)^2 + (Z_{\text{CG}} - z_i)^2}.$$

Once we compute ${d_i}^{cg}$, we sort these distances and create a distance vector ${V_1}^{cg}$ for the point set $S_1$. Similarly, we create a vector ${V_2}^{cg}$ for $S_2$ and compare whether ${V_1}^{cg}$ and ${V_2}^{cg}$ are the same. If the distance vectors are the same, then we find the convex hulls of the point sets and check whether the hulls are the same. This can be done in $O(n \log n)$ time, and hence, the entire algorithm runs in $O(n \log n)$ time.

Theorem 13.1 readily yields an algorithm for structural alignment. Although in Theorem 13.1, we mentioned that we also need to find the convex hulls and check whether the hull are the same, in practice just using the sorted distance vectors from the center of gravity seems to be sufficient (see Algorithm 13.2).

**Algorithm 13.2** algorithm to check whether pointsets $S_1$ and $S_2$ are rigidly transformable

```
INPUT  : Pointsets $S_1$, $S_2$
OUTPUT: True if $S_1$ can be transformed (rigidly) to $S_2$
$(X^1, Y^1, Z^1)$ = COMPUTE_CG($S_1$);
$(X^2, Y^2, Z^2)$ = COMPUTE_CG($S_2$);
for $i \leftarrow 1$ to $n$ do
    $V^1[i] = \sqrt{(X^1 - {x^1}_i)^2 + (Y^1 - {y^1}_i)^2 + (Z^1 - {z^1}_i)^2}$;
    $V^2[i] = \sqrt{(X^2 - {x^2}_i)^2 + (Y^2 - {y^2}_i)^2 + (Z^2 - {z^2}_i)^2}$;
end
SORT($V^1$);
SORT($V^2$);
if $V^1$==$V^2$ then
    return true;
else
    return false;
end
```

### 13.4.4 Extending Theorem 13.1 for Atomic Coordinates in Protein Structure

Algorithm 13.2 returns true if an exact rigid transformation $(R, T)$ exists, which when applied to the point set $S_1$, will give $S_2$ or vice-versa. But in the context of protein structures in which there is a considerable noise while measuring the coordinates during X-ray crystallography, **exact** rigid transformations may not be meaningful. We need an algorithm that can take the coordinates of the protein substructures and determine whether one substructure can be transformed approximately into another substructure using some rigid transformation. Keeping this in mind, we extend the exact version of the algorithm based on Theorem 13.1. We define weighted distance $(W_{i,j})$ between two sorted vectors $V_i$ and $V_j$ (each of length $n$) as follows:

$$W_{i,j} = \sum_{k=1}^{n} (n-k) \times \sqrt{(V_i[k] - V_j[k])^2}$$

We also define an approximation threshold $\epsilon$ whose value is proportional to $n$. The typical value of $\epsilon$ is 1.8 for $n = 20$. We have determined the value of $\epsilon$ from several experimental runs of our program. Algorithm 13.3 incorporates these definitions, and it can detect whether two given atomic coordinate sets (from protein structures) $P_1$ and $P_2$ can be transformed approximately from one to the other, with an error of $\epsilon$. Algorithm 13.3 is much faster and simpler than the $O(n^8)$ algorithm of [13]. As our experimental data indicate, the accuracy of Algorithm 13.3 is very good. Algorithm 13.3 can be very effective in checking whether two sets of atoms have the same structure, but our main intention is to compute the local structural alignment

**Algorithm 13.3** Algorithm to check whether atomic coordinates $P_1$ and $P_2$ are approximately transformable

```
INPUT   : Pointsets $P_1$ and $P_2$; $\epsilon$
OUTPUT: True if $P_1$ can be transformed (approx) to $P_2$
$(X^1, Y^1, Z^1)$ = COMPUTE_CG($P_1$);
$(X^2, Y^2, Z^2)$ = COMPUTE_CG($P_2$);
for $i \leftarrow 1$ to $n$ do
    $V^1[i] = \sqrt{(X^1 - x^1{}_i)^2 + (Y^1 - y^1{}_i)^2 + (Z^1 - z^1{}_i)^2}$;
    $V^2[i] = \sqrt{(X^2 - x^2{}_i)^2 + (Y^2 - y^2{}_i)^2 + (Z^2 - z^2{}_i)^2}$;
end
SORT($V^1$);
SORT($V^2$);
$W_{1,2} = \sum_{k=1}^{n} (n-k) * \sqrt{(V^1[k] - V^2[k])^2}$ ;
if $W_{1,2} \leq \epsilon$ then
    return true;
else
    return false;
end
```

among the protein backbones. This is where we seek the help of our VLAFP framework presented in Section 13.3. We use Algorithm 13.3 to build a cost function VCOST$(i, j, q)$ and apply the VLAFP algorithm on top of this cost function, thus obtaining the required local structural alignment. Section 13.4.5 gives more details on building this cost function.

### 13.4.5 Building VCOST(i,j,q) Function Based on Center of Gravity

We define VCOST$(i, j, q)$ for a fragment pair of length $q$ at indices $i$ and $j$ in the protein backbones of $P_1$ and $P_2$ as follows. Let $W1_i^q = \{a_i^1, a_{i+1}^1, \ldots, a_{i+q-1}^1\}$ be the atoms in the fragment corresponding to protein $P_1$ at index $i$ in the backbone. Similarly, we can define $W2_j^q$ corresponding to protein $P_2$. The pair $(W1_i^q, W2_j^q)$ is an AFP of size $q$. Let $(\mathrm{CG1}_x, \mathrm{CG1}_y, \mathrm{CG1}_z)$ be the center of gravity for the set of atoms in $W1_i^q$ and $(\mathrm{CG2}_x, \mathrm{CG2}_y, \mathrm{CG2}_z)$ be the center of gravity for the set of atoms in $W2_j^q$. The cost function VCOST$(i, j, q)$ is defined as follows:

$$\begin{aligned}
d_k^1 &= (x_k^1 - \mathrm{CG1}_x)^2 + (y_k^1 - \mathrm{CG1}_y)^2 + (z_k^1 - \mathrm{CG1}_z)^2, 1 \leq k \leq q \\
d_k^2 &= (x_k^2 - \mathrm{CG1}_x)^2 + (y_k^2 - \mathrm{CG1}_y)^2 + (z_k^2 - \mathrm{CG1}_z)^2, 1 \leq k \leq q \\
V^1 &= \text{Sorted distance vector of } d_k^1, 1 \leq k \leq q \\
V^2 &= \text{Sorted distance vector of } d_k^2, 1 \leq k \leq q \\
\mathrm{DAFP}(i, j, q) &= \textstyle\sum_{k=1}^{q} (V^1[k] - V^2[k])^2 \\
\mathrm{VCOST}(i, j, q) &= \frac{\mathrm{DAFP}(i, j, q)}{\sqrt{\mathrm{DAFP}(i, j, q)^2 + q^2}}
\end{aligned}$$

Once we have the normalized cost function VCOST we then can apply the VLAFP dynamic programming framework (see Algorithm 13.1) to compute the local structural alignment. Figure 13.1 illustrates the outcome of the VLAFP local alignment between 1C2N and 1COT PDB structures based on the center of gravity VCOST function. Also Figure 13.2 displays the local alignment between 1HIJ and IITI. The local alignments are marked in red. We refer to the combination of the center-of-gravity-based VCOST function with VLAFP framework as structural alignment based on center of gravity (SACG).

## 13.5 SEARCHING STRUCTURAL MOTIFS

Finding structural patterns among the protein structures is of immense interest for biologists, who often look for structural patterns including the side-chain conformations [9]. No existing algorithms addresses this issue of identifying structurally similar patterns including the side-chain conformations. Biologists often want to search for a part of the protein structure (substructure) in the existing proteins in the PDB. Finding similar substructures including the side chains is a more difficult problem

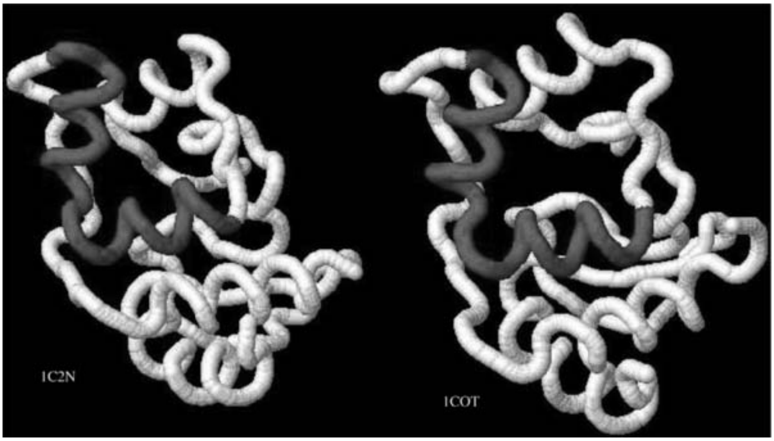


**Figure 13.1** Local structural alignment between 1C2N and 1COT using our SACG algorithm.

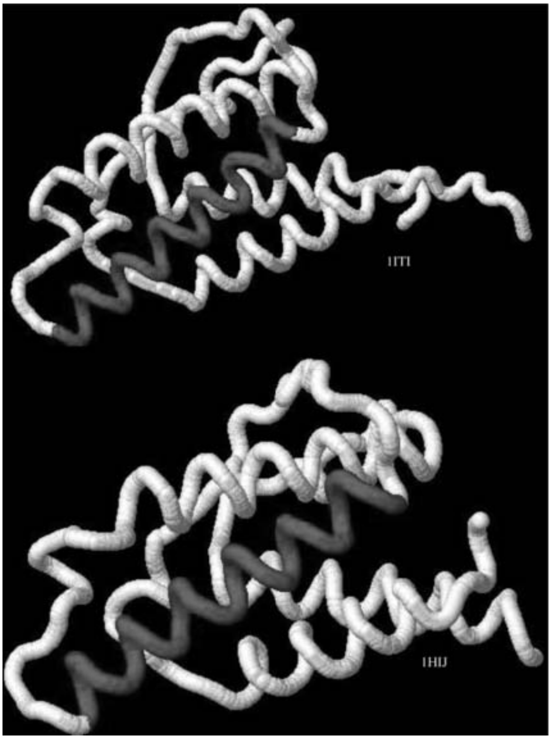
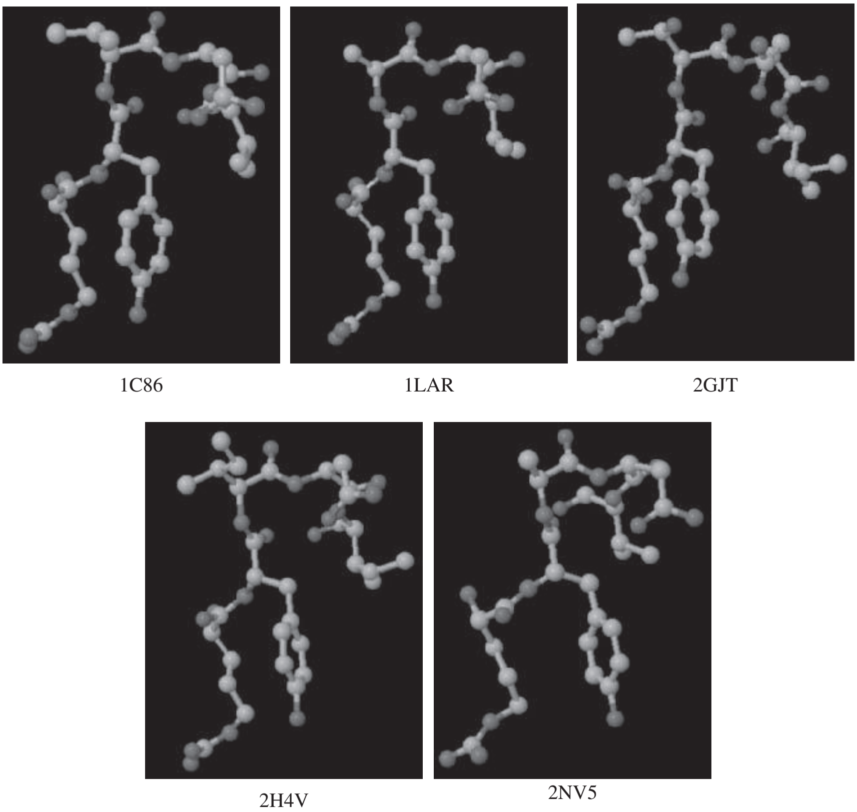


**Figure 13.2** Local structural alignment between 1HIJ and IITI using our SACG algorithm.

because of the ordering of the atoms on the side chains is not necessarily fixed. The ordering of side chain atoms in the PDB file for the same structure can vary from experiment to experiment.

Our algorithms to identify similar substructures easily can address this ordering issue because we use the sorted distances from the center of gravity as a signature to identify the substructure. The problem with different ordering of the atoms will not affect our algorithm. Algorithm 13.3 can be used readily to identify structural motifs including the side chains. Biologists can supply the list of 3-D coordinates of the atoms (in any order) to Algorithm 13.3. The algorithm then creates a sorted distance vector ($d_{\text{sig}}$) for that set of 3-D coordinates and search the entire PDB database to identify the regions that have signatures similar to $d_{\text{sig}}$. All regions that have a signature close to $d_{\text{sig}}$ can be potential structural motifs. Figure 13.3 shows the real substructures (Tyrosine phosphorylated substrates [9]) found by Algorithm 13.3. We have got these regions by taking a subset of atomic coordinates from a known YXN motif and searched the entire PDB database for regions having a signature similar to the atoms in YXN motif and identified the regions in 1C86, 1LAR, 2H4V, 2GJT, and 2NV5, as shown in Figure 13.3.



1C86

1LAR

2GJT

2H4V

2NV5

**Figure 13.3** Tyrosine phosphorylated substrates (XYXNX motifs) identifed by Algorithm 13.3 in 1C86, 1LAR, 2H4V, 2GJT, and 2NV5.

## 13.6 USING SACG ALGORITHM FOR CLASSIFICATION OF NEW PROTEIN STRUCTURES

An important problem in structural alignment is to predict accurately protein structures from the PDB that are close to a newly discovered protein structure $P_{\text{new}}$. This can be addressed easily by computing all pairwise local structural alignments between the new protein $P_{\text{new}}$ and existing proteins in the PDB database and by ranking all alignments according to the length of the local structural alignment and the normalized costs. We have used our VLAFP framework to perform the local structural alignments and ranked the proteins based on length (number of residues) and the cost(VCOST) of the alignment.

## 13.7 EXPERIMENTAL RESULTS

We provide two sets of experimental data. The first set covers all experimental data related to the classification accuracy of the SACG algorithm, and the second set contains the experimental data related to the structural motif search. The algorithm was implemented in C, and the entire source code and all datasets/results can be downloaded from `http://trinity.engr.uconn.edu/~vamsik/VAFP_ALGO/`. The program was run on a 1GB (RAM), 1.3 GHZ intel processor linux machine.

## 13.8 ACCURACY RESULTS

Our dataset is the same standard dataset used by PSIST [3] and other algorithms like Progress and geometric hashing. Please see [3] for additional details of the dataset. The dataset consisted of 181 superfamilies, and each of the superfamilies had at least 10 protein structures. The proteins are chosen in such a way that there is less than 30% of sequence homology between any two proteins from the same superfamily. The superfamilies are based on structural classification of proteins (SCOP) [2] classification. So our database consists of around 2000 proteins. The query sample is a sample of 176 proteins selected randomly from these 2000 proteins. PSIST used the same sample size. Once the sample is selected, we run our algorithm (SACG) and PSIST and classify the results based on the most frequently occuring superfamily and class in the top 20 ranked proteins. The results indicate that our algorithm achieves an average accuracy of 84.09% (superfamily) and 86.93 % (class). See Table 13.1 for additional details. Table 13.2 and Table 13.3 show the results of the top-ranked proteins for query proteins 1c2n and 1hsm using our algorithm.

### Experimental results in searching for structural motifs

Now we illustrate practical results in identifying a functional structural motif (Tyrosine phosphorylated substrate) in some PDB structures. We started with 1C86, which has a functional motif between atoms (348 and 392) (please refer to the PDB file of protein 1C86). We make the atom list from 348 to 392 in 1C86 as $S_1$ and apply our

**Table 13.1 Accuracy comparison between PSIST and SACG**

| Algorithm | Correct (SF) | Correct (Class) | Top-K | Accuracy (SF) | Accuracy (Class) |
|---|---|---|---|---|---|
| PSIST | 120 | 129 | K = 20 | 68.18% | 73.29% |
| CG_ALGO | 148 | 153 | K = 20 | 84.09% | 86.93% |

Algorithm 13.3. We found that 1LAR, 2GJT, 2NV5, and 2H4V (see Table 13.4) have highly similar substructures (Tyrosine phosphorylated substrate) to the one in 1C86 between atoms 348 and 392. Please see Table 13.3 for the actual locations of this in the PDB files. Please refer to Figure 13.3 for 3-D-visualization of these substructures.

## 13.9 CONCLUSION

In this chapter, we have introduced a new idea of using variable length alignment fragment pairs in performing local structural alignment among the proteins. We also showed how to use the VLAFP framework to classify proteins and search for structural motifs. In addition, we have introduced a new scoring function based on center

**Table 13.2 Top scored proteins for query pdb1c2n sf(46626) cl(46456) with SACG. '*c*' indicates that the class of query matches the class of the corresponding protein**

| Match | Length | Cost | pdb-id | Superfamily (sf) | Class (cl) |
|---|---|---|---|---|---|
| *c* | 44 | 24.13 | pdb1mbj- | 46689 | 46456 |
| *c* | 34 | 21.99 | pdb2bby- | 46785 | 46456 |
| *c* | 40 | 26.39 | pdb1jtb- | 47699 | 46456 |
| *c* | 46 | 31.96 | pdb1hsn- | 47095 | 46456 |
| *c* | 36 | 25.84 | pdb1nhm- | 47095 | 46456 |
| *c* | 32 | 23.05 | pdb1mbe- | 46689 | 46456 |
|  | 37 | 26.80 | pdb2cjo- | 54292 | 53931 |
| *c* | 35 | 25.40 | pdb1uxd- | 47413 | 46456 |
| *c* | 36 | 26.39 | pdb1aab- | 47095 | 46456 |
| *c* | 39 | 28.69 | pdb1mbk- | 46689 | 46456 |
|  | 40 | 29.54 | pdb1eot- | 54117 | 53931 |
| *c* | 37 | 27.86 | pdb1etd- | 46785 | 46456 |
| *c* | 46 | 35.43 | pdb1nhn- | 47095 | 46456 |
|  | 47 | 36.75 | pdb1e09-A | 55961 | 53931 |
| *c* | 47 | 38.05 | pdb2new- | 48695 | 46456 |
|  | 45 | 36.70 | pdb1bt7- | 50494 | 48724 |
| *c* | 50 | 41.15 | pdb1gjt-A | 46997 | 46456 |
|  | 32 | 26.90 | pdb4ull- | 50203 | 48724 |
| *c* | 37 | 31.49 | pdb1a2i- | 48695 | 46456 |
| *c* | 47 | 40.16 | pdb1wjd-B | 46919 | 46456 |
| *c* | 47 | 40.57 | pdb1wjd-A | 46919 | 46456 |

+ve **cl** classification (46456) occurs 16 times, −ve **sf** classification (47095) occurs four times.

**Table 13.3 Top scored proteins for query pdb1hsm sf(47095) cl(46456) using SACG. '*c*sf' indicates both class and superfamily of the query matches with the corresponding protein**

| Match | Length | Cost | pdb-id | Superfamily (sf) | Class (cl) |
|---|---|---|---|---|---|
| *c*sf | 168 | 72.65 | pdb1hsn- | 47095 | 46456 |
| *c* | 33 | 19.64 | pdb2bby- | 46785 | 46456 |
| | 31 | 19.40 | pdb2cjo- | 54292 | 53931 |
| *c*sf | 145 | 95.48 | pdb1nhm- | 47095 | 46456 |
| *c* | 38 | 26.18 | pdb1ba5- | 46689 | 46456 |
| *c* | 38 | 28.57 | pdb1edj- | 46997 | 46456 |
| *c* | 39 | 30.66 | pdb1wtu-B | 47729 | 46456 |
| *c*sf | 127 | 102.17 | pdb1nhn- | 47095 | 46456 |
| *c*sf | 107 | 86.43 | pdb1hmf- | 47095 | 46456 |
| *c*sf | 110 | 89.27 | pdb1hme- | 47095 | 46456 |
| | 33 | 27.14 | pdb1bc6- | 54862 | 53931 |
| | 35 | 29.19 | pdb1grx- | 52833 | 51349 |
| | 42 | 35.98 | pdb2cjn- | 54292 | 53931 |
| *c* | 37 | 31.73 | pdb1tnt- | 46785 | 46456 |
| | 41 | 35.81 | pdb1mit- | 54654 | 53931 |
| *c*sf | 52 | 46.02 | pdb1hma- | 47095 | 46456 |
| *c* | 36 | 32.18 | pdb1bqv- | 47769 | 46456 |
| *c* | 46 | 41.35 | pdb1hue-A | 47729 | 46456 |
| *c* | 42 | 38.25 | pdb1mbg- | 46689 | 46456 |
| *c* | 35 | 31.92 | pdb1bdc- | 46997 | 46456 |
| | 33 | 30.16 | pdb1svq- | 55753 | 53931 |

+ve **cl** classification (46456) occurs 15 times, +ve **sf** classification (47095) occurs six times.

**Table 13.4 Regions having tyrosine phosphorylated substrate found by Algorithm 13.3**

| Protein(PDB-ID) | Region(start-end) |
|---|---|
| 1LAR | Residue 383 to 425 |
| 2GJT | Residue 427 to 472 |
| 2NV5 | Residue 430 to 470 |
| 2H4V | Residue 440 to 486 |

of gravity. Experimental results indicate that using the VLAFP framework can produce better local structural alignments compared with using constant size AFPs.

## ACKNOWLEDGMENTS

This research has been supported in part by NIH Grant 1R01GMO79689-01A1, NSF Grant ITR-0326155 and NSF Grant 0829916.

## REFERENCES

1. H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Res*, 28:235–242, 2000.
2. A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol*, 247:536–540, 1995.
3. F. Gao and M.J. Zaki, PSIST: indexing protein structures using suffix trees. *Proceedings of the IEEE Computational Systems Bioinformatics Conference*, Standford, CA, 2005.
4. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, 1990.
5. G.H. Cohen. ALIGN: A program to superimpose protein coordinates, accounting for insertions and deletions. *J Appl Crystallogr*, 1997.
6. M. Gerstein and M. Levitt. Using iterative dynamic programming to obtain accurate pairwise and multiple alignments of protein structures. *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, Menlo Park, CA, 1996, pp. 59–67.
7. L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *J Mol Biol*, 233:123–138, 1993.
8. I.N. Shindyalov and P.E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Eng* 11(9):739–747, 1998.
9. K.H. Kirsch, M. Kensinger, and H. Hanafusa. A p130Cas tyrosine phosphorylated substrate domain decoy disrupts v-Crk signaling. *BMC Cell Biol*, 2002.
10. M.J. Betts, G. Agarwal, and R.B. Russell. Exon structure conservation despite low sequence similarity: a relic of dramatic events in evolution?. *EMBO J*, 20(19):5354–5360, 2001.
11. M.D. Atkinson. An optimal algorithm for geometric congruence. *J Algorithm*, 8:159–172, 1987.
12. M.J. Atallah. Checking similarity of planar figures. *Int J Comput Inform Sci*, 13:279–290, 1984.
13. H. Alt, K. Melhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. *Discrete Comput Geom*, 3:237–256, 1988.
14. T. Akutsu. Protein structure alignment using dynamic programming and iterative improvement. *IEICE Trans Info Syst*, E78–D(0):1996.
15. T. Akutsu. Algorithms for determining the geometrical congruity in tow and three dimensions. *Proceedings of the 3rd. International Symposium on Algorithms and Computation*, Nagoya, Japan, 1992.
16. Y. Zhang and J. Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins*, 57:702–710, 2004.
17. Y. Zhang and J. Skolnick. TM-align: A protein structure alignment algorithm based on TM-score. *Nucleic Acids Res*, 33:2302–2309, 2005.

# 14

# EVOLUTION OF THE CLUSTAL FAMILY OF MULTIPLE SEQUENCE ALIGNMENT PROGRAMS

Mohamed Radhouene Aniba and Julie Thompson

## 14.1 INTRODUCTION

One of the cornerstones of modern bioinformatics is the comparison or alignment of protein sequences. Multiple alignments are used to compare a set of sequences, either to estimate their overall similarity or to identify locally conserved motifs. As such, classical string-matching algorithms have been applied to the problem as well as many other algorithms, such as dynamic programming, hidden Markov Models, genetic algorithms, and so on. Nevertheless, it is important to keep in mind that the sequences analyzed in biology represent biological molecules (DNA, RNA, or protein) having unique three-dimensional (3-D) structures and specific functions. They act in complex networks, interacting with other molecules in a stable or transitory way within a changing cellular environment.

By placing the sequence in the framework of the overall family, multiple alignments not only identify important structural or functional motifs that have been conserved during evolution but also can highlight particular nonconserved features resulting from specific events or perturbations [32, 15]. As a consequence, multiple sequence comparison or alignment has become a fundamental tool in many different domains in modern molecular biology, from evolutionary studies to prediction of

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

two-dimensional (2-D) of 3-D structure, molecular function, intermolecular interactions, and so on.

Numerous multiple alignment programs are now available, using a wide variety of different algorithms. Many of these methods are described in detail in the chapter 12 "multiple alignment algorithms," and here we will concentrate on one specific suite of programs: the Clustal family (www.clustal.org). Clustal is the oldest of the programs still in wide use today, having been first distributed by post on floppy disks in the late 1980s. It originally was written in Microsoft Fortran for MS-DOS and ran on IBM-compatible personal computers (PCs) as four separate executable programs, Clustal1–Clustal4 [11, 12]. These programs later were rewritten in C and merged into a single program, ClustalV [13], that was distributed for VAX/VMS, Unix, Apple Macintosh, and IBM-compatible PCs.

The original Clustal was designed to perform multiple alignments of large numbers of amino acid or nucleotide sequences efficiently. The method was based on first deriving a phylogenetic tree from a matrix of all pairwise sequence similarity scores, obtained using a fast pairwise alignment algorithm. Then the multiple alignment was achieved from a series of pairwise alignments of clusters of sequences following the branching order of the tree. The method was sufficiently fast and economical with memory to be implemented easily on a microcomputer, and yet the results obtained were comparable with those from packages requiring mainframe computer facilities.

The Clustal programs in use today derive from ClustalW [27], which incorporated several enhancements to take into account the nature of the molecules under study and to facilitate the analysis of the relationships between sequence, structure, function, and evolution. These programs have been amended and added to many times since 1994 to increase functionality and to increase sensitivity. The user friendliness also was enhanced greatly by the addition, in 1997, of a graphical user interface [28]. By the late 1990s, ClustalW and ClustalX had become the most widely used multiple alignment programs. They were able to align medium-sized datasets very quickly and were easy to use. Today, ClustalW and ClustalX continue to be very widely used, increasingly on websites. The EBI Clustal site (www.ebi.ac.uk/clustalw) literally gets millions of multiple alignment jobs per year.

Development is ongoing, with the latest major release of ClustalW 2.0 and ClustalX 2.0 in 2007 [14]. Once again, the programs were completely rewritten, this time in C++ with a simple object model to make it easier to maintain the code and, more importantly, to make it easier to modify or even replace some alignment algorithms. This complete redesign now is facilitating the latest evolution of the Clustal software, away from a single isolated algorithm toward a more integrated, cooperative system combining different, complementary algorithms, and additional information other than the sequence itself.

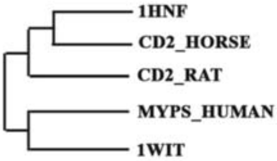
## 14.2 CLUSTAL-CLUSTALV

The first Clustal program was written by Des Higgins in 1988 [11] and was designed specifically to work efficiently on personal computers, which at that time, had feeble

**1. Do pairwise alignments and calculate a similarity matrix**

| | 1HNF | CD2_HORSE | CD2_RAT | MYPS_HUMAN | 1WIT |
|---|---|---|---|---|---|
| **1HNF** | 100 | 50 | 42 | 17 | 15 |
| **CD2_HORSE** | | 100 | 40 | 14 | 18 |
| **CD2_RAT** | | | 100 | 24 | 17 |
| **MYPS_HUMAN** | | | | 100 | 31 |
| **1WIT** | | | | | 100 |

**2. Cluster similarities and construct a "guide tree"**



**3. Do multiple alignment following the guide tree**

```
1HNF          ----TNALETWGALGQDINLDIPSFQMSDDIDDIKWEKTSDKKKIAQFRKEKETF
CD2_HORSE     GAVSKKNITILGALERDINLDIPAFQMSEHVEDIQWSK--GKTKIAKFKNGSMTF
CD2_RAT       GADCRDSGTVWGALGHGINLNIPNFQMTDDIDEVRWER--GSTLVAEFKRKMKPF
MYPS_HUMAN    ---LDADNTVTVIAGNKLRLEIP--ISGEPPPKAMWSR--GDKAIMEGS------
1WIT          PKILTASRKIKIKAGFTHNLEVDFI--GAPDPTATWTV--GDSGAALA-------
```

**Figure 14.1** The basic progressive alignment procedure in Clustal.
The algorighm is illustrated using a set of five immunoglobulin-like domains. The sequence names are from the Swissprot or protein Data Bank and are as follows (PDB) databases: IHNF: human cell adhesion (CD2) protein, CD2_HORSE: horse cell adhesion protein, CD2_RAT: rat cell adhesion protein, MYPS_HUMAN: human myosin-binding protein, and IWIT: rematode twitchin muscle protein.

computing power by today's standards. It combined a memory-efficient dynamic programming algorithm [19] with the progressive alignment strategy [6, 26], which exploits the fact that homologous sequences are related evolutionarily. A multiple sequence alignment is built up gradually using a series of pairwise alignment following the branching order in a phylogenetic tree. An example using five immunoglobulin-like domains is shown in Figure 14.1.

The next sections describe the different stages of this progressive multiple alignment approach in more detail.

### 14.2.1 Pairwise Similarity Scores

The first step involves the calculation of similarity scores for each pair of sequences to be included in the multiple alignment. The scores are calculated from fast alignments generated by the method of [33]. These are "hash" or "word" or "k-tuple" alignments carried out in three stages. First, every fragment of sequence of length k (for proteins, the default length is one residue; for DNA, it is two bases) is marked in both sequences, and all k-tuple matches between the two sequences are identified. This is similar to a dot-matrix plot between the two sequences, with each k-tuple match represented by a dot. The next step is to find those diagonals in the plot with the most matches and mark all diagonals within a specific window size of each top diagonal. This process defines the diagonal regions in the plot where the regions of similarity most likely will lie. Finally, the head-to-tail arrangement of k-tuple matches from these diagonal regions is found that gives the highest score. The score is defined as the number of exactly matching residues in this alignment, minus a "gap penalty" for every gap that was introduced.

Several options are provided that can be modified by the user to optimize the similarity score calculation depending on the set of sequences to be aligned:

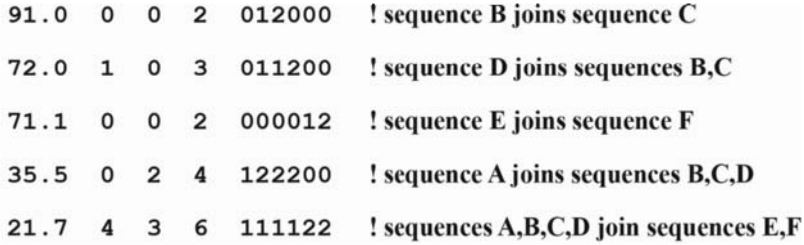
- *K-tuple size:* the length of "words" to be matched. Increasing this parameter increases the speed of the calculation, whereas decreasing it leads to improved sensitivity
- *Gap penalty:* the number of matching residues that must be found to introduce a gap
- *Number of top diagonals:* The number of best diagonals in the dot-matrix plot that are considered. Decreasing this parameter leads to increased speed, whereas increasing it improves sensitivity
- *Window size:* The number of diagonals around each "top" diagonal that are considered. Decreasing this parameter leads to increased speed, whereas increasing it improves sensitivity
- *Scoring method:* The similarity scores for each pair of sequences may be expressed as raw scores (number of identical residues minus a "gap penalty" for each gap) or as a percentage of the shorter sequence length.

### 14.2.2 Guide Tree

Once the similarity scores have been calculated for each pair of sequences, the next step is to create a "guide tree" or dendogram, which will be used to determine the order of alignment in the final progressive alignment step.

The original Clustal and ClustalV programs used a simple bottom-up data clustering method to build the tree, known as the unweighted pair grouping method with arithmetic means (UPGMA) [25]. This is a form of cluster analysis, and the end result is a representation of the sequence similarity as a hierarchy. An example for six sequences is shown in Figure 14.2.

Each row in the dendogram represents the joining together, or grouping, of two or more sequences. For N sequences, there are N-1 groupings; hence, there are five rows in Figure 14.2 for six sequences. The first number in each row is the similarity score for the grouping. The last six digits in the line show which sequences are grouped, where each sequence is represented by one digit. At each step, all sequences with the digit "1" are joined to all sequences with digit "2" (sequences with digit "0" are not excluded). Thus, the hierarchy progresses from the top down, joining more and



```
91.0  0  0  2  012000   ! sequence B joins sequence C
72.0  1  0  3  011200   ! sequence D joins sequences B,C
71.1  0  0  2  000012   ! sequence E joins sequence F
35.5  0  2  4  122200   ! sequence A joins sequences B,C,D
21.7  4  3  6  111122   ! sequences A,B,C,D join sequences E,F
```

**Figure 14.2** An example dendogram is used as a guide tree in Clustal.
An example dendogram is shown for six sequences, named A-F. The meaning of the different columns is explained in the text.

more sequences until all are joined together. For example, in the first row, sequence B joins sequence C at a similarity level of 91% identity; next, sequence D joins the previous grouping of B plus C at a level of 72%, and so on. The other three columns of numbers are a pointer to the row from which the "1" sequences last were joined (or zero if there is only one of them), a pointer to the row in which the "2" sequences last were joined, and the total number of sequences joined in this line.

### 14.2.3 Progressive Multiple Alignment

The final stage in the alignment progress is the progressive multiple alignment of all sequences. First, the two sequences with the largest similarity score are aligned. Then, all sequences are aligned gradually from the closest to the most distant following the order of grouping specified in the dendogram.

### 14.2.4 An Efficient Dynamic Programming Algorithm

At each alignment stage (corresponding to a row in the dendogram), a dynamic programming algorithm is used to align two sequences, a single sequence with a group of sequences, or two groups of sequences. The dynamic programming algorithm for aligning two sequences S1 and S2, with residues $\mathrm{a}_1,\ldots,\mathrm{a}_\mathrm{M}$ and $\mathrm{b}_1,\ldots,\mathrm{b}_\mathrm{N}$, respectively, optimizes a so-called sum-of-pairs score and can be summarized by the following recursion:

$$H_{i,j} = \left\{ \begin{array}{c} H_{i-1,j-1} + C_{i,j} \\ \max\left\{H_{i-k,j} - (\mathrm{GOP} + k \times \mathrm{GEP})\right\} \\ \max\left\{H_{i,j-l} - (\mathrm{GOP} + l \times \mathrm{GEP})\right\} \end{array} \right\}$$

where $\mathrm{C}_{i,j}$ is the comparison matrix score for aligning residues $\mathrm{a}_i$ and $\mathrm{b}_j$, GOP is the gap opening penalty, and GEP is the gap extension penalty for extending a gap by one residue. The residue comparison matrix and gap penalties are described in more detail subsequently. The alignment of two groups of sequences (or profiles) is a simple extension of the algorithm in which the score for aligning two positions is simply the average score for aligning each pair of residues in the respective profiles.

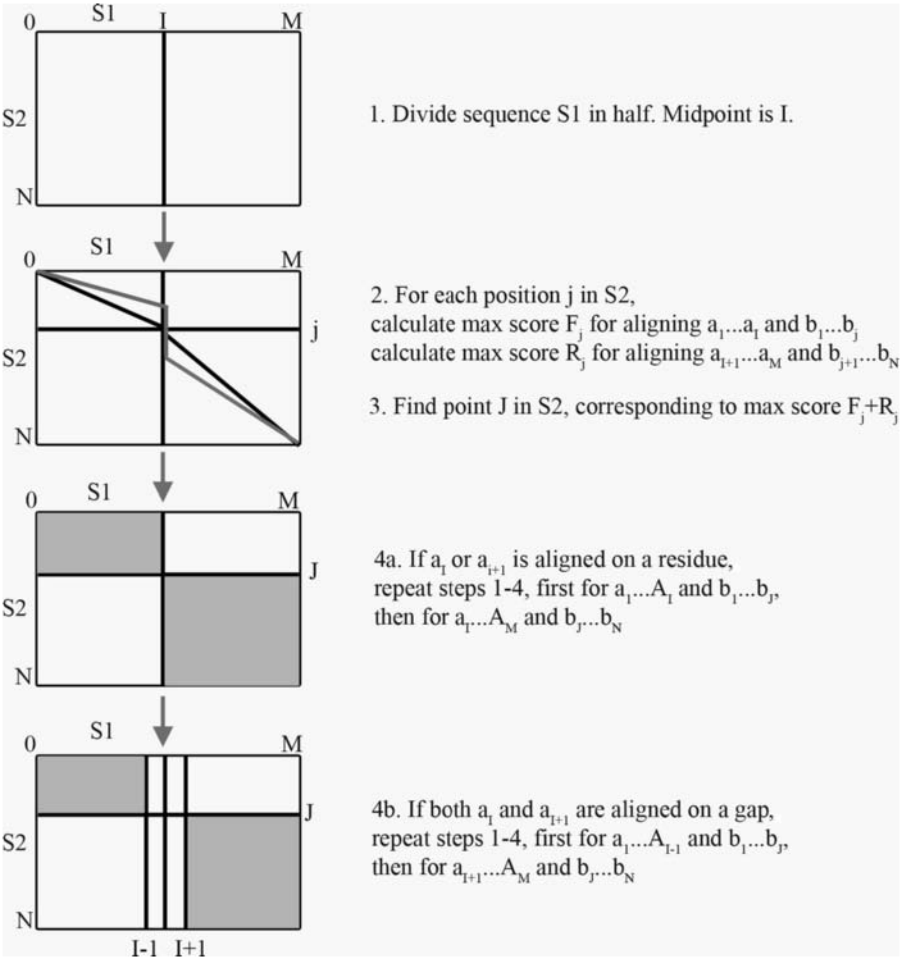

0
S1
I
M
S2
N
1. Divide sequence S1 in half. Midpoint is I.
0
S1
M
j
S2
N
2. For each position j in S2,
calculate max score $F_j$ for aligning $a_1...a_I$ and $b_1...b_j$
calculate max score $R_j$ for aligning $a_{I+1}...a_M$ and $b_{j+1}...b_N$
3. Find point J in S2, corresponding to max score $F_j+R_j$
0
S1
M
J
S2
N
4a. If $a_I$ or $a_{I+1}$ is aligned on a residue,
repeat steps 1-4, first for $a_1...A_I$ and $b_1...b_J$,
then for $a_I...A_M$ and $b_J...b_N$
0
S1
M
J
S2
N
I-1
I+1
4b. If both $a_I$ and $a_{I+1}$ are aligned on a gap,
repeat steps 1-4, first for $a_1...A_{I-1}$ and $b_1...b_J$,
then for $a_{I+1}...A_M$ and $b_J...b_N$

**Figure 14.3** Schematic outline of the Myers and Miller algorithm.
Steps involved in finding an optimal alignment between two sequences, S1 and S2. Sequence S1 is of length M and has residues denoted $a_1,\ldots a_M$. Sequence S2 is of length N and has residues $b_1,\ldots b_N$. Steps 1–4 are repeated until all residues have been aligned.

To calculate the optimal alignment in linear time and space, Clustal uses the algorithm of [19], which is a very memory efficient variation of Gotoh's algorithm [8]. This recursive algorithm delivers an optimal alignment by first dividing sequence S1 in half then finding the best residue in sequence S2 to align with this midpoint. Pairs of residues are found recursively on both sides of the midpoint, until all residues in sequence S1 are aligned. Figure 14.3 provides an outline of the algorithm for the alignment of two sequences.

The Myers and Millers algorithm calculates an optimal alignment with space requirements of O(M + N) and time O(MN), where M and N are the lengths of the two sequences. The alignment is considered "optimal" if it gives the best sum-of-pairs score for the aligned residues depending on the parameters used. Unfortunately, for any given set of sequences, the best parameters that will give a biologically "correct" alignment are generally unknown, and in practice, it is normally necessary to test a certain number of parameter combinations. Clustal provides menu options that can be selected by the user to modify the most important parameters, namely the residue comparison matrix and the gap opening and extension penalties:

- *Residue comparison matrix:* The simplest matrix assigns a score for each pair of identical residues that are aligned. This is known as the identity matrix. For example, for alignments of nucleotide sequences, the identity matrix would assign the same score to a match of the four classes of bases, ACGT, and 0 for any mismatch. When the sequences to be aligned are closely related, this usually will find approximately the correct solution. However, for more divergent sequences (sharing less than 25–30 % identity), the scores given to nonidentical, but similar, residues become more important. Therefore, more sophisticated matrices have been developed for both DNA and protein sequences. These matrices can be stored in a text file and input to Clustal if required. To continue the nucleotide sequence example, transitions (substitution of A-G or C-T) happen much more frequently than transversions (substitution of A-T or G-C), and it is often desirable to score these substitutions differently. More complex matrices also exist in which matches between ambiguous nucleotides are scored whenever there is any overlap in the sets of nucleotides represented by the two symbols being compared. For protein sequences, the matrices generally take into account the biochemical similarities between residues and/or the relative frequencies with which each amino acid is substituted by another. Widely used matrices include: the point accepted mutation (PAM) matrices [34], the Blosum matrices [10], or the Gonnet matrices [4]. Other more specialized matrices also have been developed (*e.g.*, for specific secondary structure elements [16] or for the comparison of particular types of proteins, such as transmembrane proteins [21].
- *Gap opening and extension penalties:* One of the first gap scoring schemes for the alignment of two sequences charged a fixed penalty for each residue in either sequence aligned with a gap in the other. Thus, the cost of a gap is proportional to its length. Alignment algorithms implementing such length-proportional gap penalties are efficient, although the resulting alignments often contain numerous short insertions or deletions that are not biologically meaningful. To address this problem, Clustal uses linear or "affine" gap costs [1] that define a gap insertion or "gap opening" penalty in addition to the length-dependent or "gap extension" penalty. The goal is to mimic the biological processes or constraints that are thought to regulate the evolution of DNA or protein sequences. Thus, a smaller number of long gaps is preferred over many short ones. Fortunately, algorithms using affine gap costs are only slightly more complex than those using length-proportional gap penalties, requiring only a constant factor more space and time.

### 14.2.5 Profile Alignments

As mentioned earlier, the progressive algorithm gradually builds up a multiple alignment by aligning larger and larger groups of sequences. At each step, a pairwise alignment is performed involving two sequences, a single sequence with a group of sequences, or two groups of sequences. The alignment of groups of sequences (otherwise known as profiles) is achieved by an extension of the algorithm developed by [9]. Profile alignments are a simple extension of sequence–sequence alignments in which the score for matching residues is replaced by a score for matching positions in the profiles. This score is defined as the mean comparison matrix score of all residues in one alignment versus all those in the other. For example, if we consider two alignments with M and N sequences, respectively, then the score at any position is the average of the $M \times N$ scores of the residues compared separately. Any gaps that are introduced are inserted in all sequences in the alignment at the same position.

This alignment strategy works well when the sequences to be aligned are of different degrees of divergence. Pairwise alignment of closely related sequences can be performed very accurately. By the time the more distantly related sequences are aligned, important information about the variability at each position is available from those sequences already aligned. Unfortunately, this greedy approach works less well when the sequences do not have a smooth evolutionary distribution or when all sequences are very divergent.

## 14.3 CLUSTALW

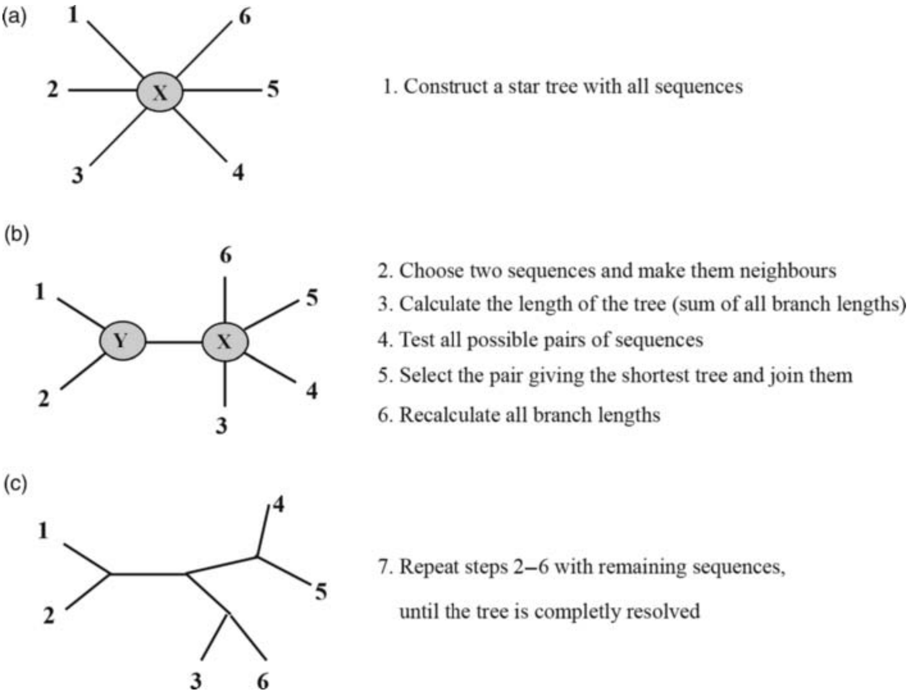
The most important risks regarding the progressive multiple alignment are, first, the choice of inappropriate alignment parameters and, second, the possibility that a wrong alignment at the initial stages may be amplified in subsequent steps. These problems led to the development of a new version of the program, ClustalW [27], which introduced several improvements to minimize the risks.

### 14.3.1 Optimal Pairwise Alignments

The initial pairwise sequence comparisons are performed using an optimal dynamic programming algorithm [20] with a residue comparison matrix and two gap penalties (for opening or extending gaps) rather than the more approximative k-tuple method used originally. The pairwise similarity scores then are defined as the percentage of identities in the optimal alignment compared with the number of residues aligned (gap positions are excluded).

### 14.3.2 More Accurate Guide Tree

The guide tree in ClustalW is built using another phylogenetic tree construction method called neighbour-joining (NJ) [24]. The algorithm is shown briefly in Figure 14.4.



1. Construct a star tree with all sequences

2. Choose two sequences and make them neighbours
3. Calculate the length of the tree (sum of all branch lengths)
4. Test all possible pairs of sequences
5. Select the pair giving the shortest tree and join them
6. Recalculate all branch lengths

7. Repeat steps 2–6 with remaining sequences,
until the tree is completly resolved

**Figure 14.4** Outline of the neighbour-joining algorithm.
The construction of the tree is shown for six sequences, numbered 1–6. X and Y represent potential nodes in the tree. The initial steps in the construction are shown in (a) and (b). The tree shown in (c) is the final neighbour-joining tree.

Although the NJ method is less efficient than the UPGMA method used in the original Clustal version, it has been tested extensively and usually finds a tree that is close to the optimal tree. As input, the method requires pairwise sequence distance scores rather than similarities. Therefore, the percentage similarity scores calculated, are converted to distances by dividing by 100 and subtracting from 1.0 to give the number of differences per site.

### 14.3.3 Improved Progressive Alignment

***14.3.3.1 Automatic Parameter Choice.*** During the progressive multiple alignment, different residue comparison matrices are used depending on the divergence of the sequences to be aligned. Some matrices are more appropriate for aligning closely related sequences, whereas others work better at greater evolutionary distances. In ClustalW, a matrix is selected automatically depending on the distance between the two sequences or groups of sequences to be compared. These distances are obtained directly from the guide tree. In addition, suitable gap opening and extension penalties are selected automatically. Initially, two gap penalties are used: a GOP, which gives the cost of opening a new gap of any length, and a GEP, which gives the cost of every item in a gap. The software then automatically attempts to choose appropriate gap penalties for each sequence alignment, depending on the following factors:

– *Dependence on the weight matrix.* It has been shown [13] that varying the gap penalties used with different weight matrices can improve the accuracy of sequence alignments. ClustalW uses the average score for two mismatched residues (*i.e.*, off-diagonal values in the matrix) as a scaling factor for the GOP.

– *Dependence on the similarity of the sequences.* First, the percent identity of the two (groups of) sequences to be aligned is estimated from the pairwise alignment. This then is used to increase the GOP for closely related sequences or to decrease it for more divergent sequences on a linear scale.

– *Dependence on the lengths of the sequences.* The scores for sequence alignments increase with the length of the sequences whether the alignment is optimal or not. Therefore, we use the logarithm of the length of the shorter sequence to increase the GOP with sequence length. Using these three modifications, the initial GOP calculated by the program is:

> GOP – [GOP + log[min(N,M)]} × (average residue mismatch score) × (per cent identity scaling factor), where N and M are the lengths of the two sequences.

– *Dependence on the difference in the lengths of the sequences.* The GEP is modified depending on the difference between the lengths of the two sequences to be aligned. If one sequence is much shorter than the other, then the GEP is increased to inhibit too many long gaps in the shorter sequence. The initial GEP calculated by ClustalW is:

> GEP – GEP × [1.0 + log(N/M)], where N and M are the lengths of the two sequences.

***14.3.3.2 Position-Specific Gap Penalties.*** In most dynamic programming applications, the initial gap opening and extension penalties are applied equally at every position in the sequence, regardless of the location of a gap, except for terminal gaps, which usually are allowed at no cost. In ClustalW, position-specific gap penalties are incorporated to encourage the opening of gaps at specific positions (Figure 14.5), such as regions already containing gaps (often corresponding to loops in the 3-D structure) rather than conserved regions (such as regular secondary structures).

The local gap penalty modification rules are applied in a hierarchical manner. The exact details of each rule are given below. First, if there is a gap at a position, then the gap opening and gap extension penalties are lowered; the other rules do not apply. This makes gaps more likely at positions where there are already gaps. If there is no gap at a position, then the gap opening penalty is increased if the

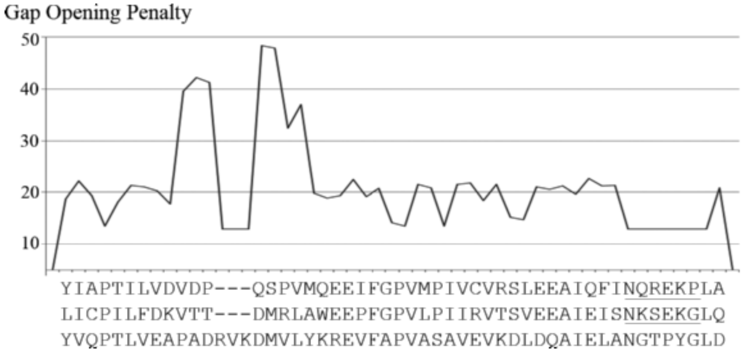


**Figure 14.5** Calculation of variable gap penalties.
The gap opening penalties are plotted for an example alignment. Two sequence segments with hydrophilic residues are underlined. The lowest penalties are observed at the ends of the alignment, the hydrophilic segments, and the three positions with gaps in the alignment. The highest values correspond to the positions to each side of the two gaps. The remaining variation is caused by the residue-specific gap penalties.

position is within eight residues of an existing gap. This discourages gaps that are too close together. Finally, at any position within a run of hydrophilic residues, then the penalty is decreased. These runs usually indicate loop regions in protein structures. If there is no run of hydrophilic residues, then the penalty is modified using a table of residue-specific gap propensities (12). These propensities were derived by counting the frequency of each residue at either end of the gaps in alignments of proteins of a known structure (Figure 14.5).

- *Lowered gap penalties at existing gaps*. If there are already gaps at a position, then the GOP is reduced in proportion to the number of sequences with a gap at this position, and the GEP is lowered by a half. The new gap opening penalty is calculated as:

$$\text{GOP} - \text{GOP} \times 0.3 \times (\text{no. of sequences without a gap/no. of sequences}).$$

- *Increased gap penalties near existing gaps*. If a position does not have any gaps but is within eight residues of an existing gap, then the GOP is increased by:

$$\text{GOP} - \text{GOP} \times t2 + [(8 - \text{distance from gap}) \times 2]/8j$$

- *Reduced gap penalties in hydrophilic stretches*. Any run of five hydrophilic residues is considered to be a hydrophilic stretch. The residues that are to be considered hydrophilic may be set by the user but are conservatively set to D, E, G, K, N, Q, P, R, or S by default. If, at any position, there are no gaps and any of the sequences has such a stretch, then the GOP is reduced by one third.
- *Residue-specific penalties*. If there is no hydrophilic stretch and the position does not contain any gaps, then the GOP is modified by the propensity of the residues at this position to be found next to gaps [22].

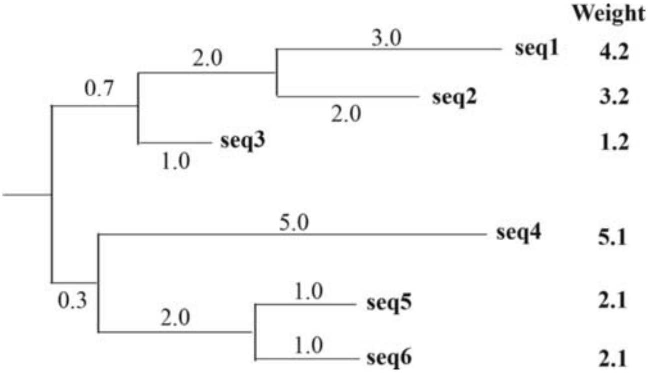


**Figure 14.6** Sequence weight calculation.
The sequence weights are calculated based on the branch lengths in the guide tree. The branches along the path from a sequence to the root are said to be owned by the sequence, and the order of a branch is defined by the number of sequences that own the branch. The sequence weight then is defined as the sum of the branch lengths divided by the order of the branch. For example, the weight for sequence "seq1" is (3.0/1 + 2.0/2 + 0.7/3) = 4.2.

***14.3.3.3 Weighted Profile Alignments.*** As described, to align two groups of sequences (profiles), we need to calculate a score for matching each position in the first profile with each position in the second profile. To correct for unequal sampling of sequences, weights are introduced into the profile alignments. Suppose we have two profiles, P1 and P2, with M and N sequences, respectively. The score for aligning any position in P1 with another position in P2 is defined as follows:

$$\frac{\sum_{i=0}^{i=M} \sum_{j=0}^{j=N} C(a_i, b_j) \times W_i \times W_j}{M \times N}$$

where $C$ is the comparison matrix score, $a_i$ is the residue in the $i$-th sequence in P1, $b_j$ is the residue in the $j$-th sequence in P2, and $W_i$ is weight for sequence $i$.

The weighting scheme is designed to down-weight closely related sequences and to up-weight the most divergent ones. The weights are calculated based on the lengths of the branches in the guide tree (Figure 14.6).

***14.3.3.4 Divergent Sequences.*** The most divergent sequences (most different on average from all other sequences) are usually the most difficult to align correctly. It is sometimes better to delay the incorporation of these sequences until all of the more easily aligned sequences are merged first. This may give a better chance of correctly placing the gaps and matching weakly conserved positions against the rest of the sequences. A choice is offered to set a cut off (default is 30% identity or less with any other sequence) that will delay the alignment of the divergent sequences until all of the rest have been aligned.

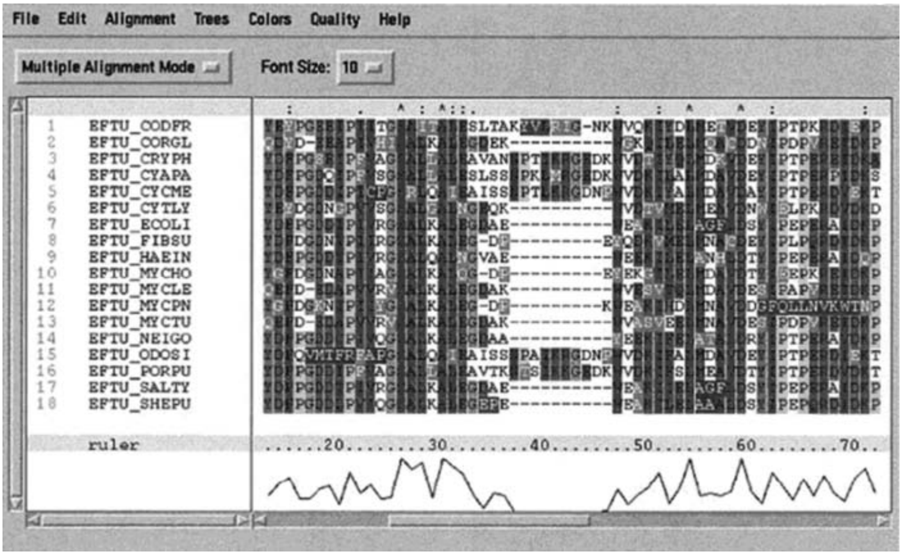


**Figure 14.7** ClustalX graphical user interface.
Residues are colored according to the conservation observed at each position. Conservation scores are plotted below the alignment. A high score indicates a highly conserved column, a low score indicating a less well-conserved position. The white residues on a grey or black background indicate the results of alignment quality analyses (see main text).

## 14.4 CLUSTALX

In the 1980s, a computer revolution occurred with the appearance of new operating systems (*e.g.*, MAC OS, Windows Microsoft, or UNIX X Windows) that had windowing capacities and a graphical user interface (GUI). These systems made it much easier to access information by exploiting graphical images in addition to simple text. Clustal also profited from these new developments with the development of a new user-friendly interface, ClustalX [28]. The software included novel algorithms for alignment quality analysis and flexible strategies for the correction of the initial alignment. Despite of the improvements introduced in ClustalW, a manual refinement was still necessary for complex alignment problems to obtain a high-quality alignment.

ClustalX displays the sequence alignment in a window (Figure 14.7) and a color coding of the residues as well as the conservation scores plotted below the alignment are used to highlight motifs or conserved features. Pull-down menus provide access to all options necessary at each stage of the multiple sequence alignment. Furthermore, facilities are provided that allow the user to build up manually a multiple alignment in difficult cases, using successive sequence–profile or profile–profile alignments.

### 14.4.1 Alignment Quality Analysis

Several methods are provided to analyze the quality of an alignment and to identify low-scoring regions and outlier residues. These methods, in combination with the facilities to realign specific regions of the alignment, provide flexible tools to detect errors in the input sequences and the refinement of the multiple alignment.

***14.4.1.1 Conservation Scores.*** The conservation scores plotted below the alignment in the ClustalX window are calculated using an analogous approach to that introduced by [31], who used a geometric analysis based on a continuous sequence space to compare sequence weighting methods. Suppose we have an alignment of M sequences of length N. Then, the alignment can be written as follows:

$$\begin{array}{l}
\mathrm{A}_{1,1}\ \mathrm{A}_{1,2}\ \mathrm{A}_{1,3}\ldots\ldots\ldots\ldots\mathrm{A}_{1,N}\\
\mathrm{A}_{2,1}\ \mathrm{A}_{2,2}\ \mathrm{A}_{2,3}\ldots\ldots\ldots\ldots\mathrm{A}_{2,N}\\
.\\
.\\
\mathrm{A}_{M,1}\mathrm{A}_{M,2}\mathrm{A}_{M,3}\ldots\ldots\ldots\mathrm{A}_{M,N}
\end{array}$$

Suppose we also define a residue comparison matrix $C$ of size $R \times R$, where $R$ is the number of residues. $C(a, b)$ is the score for aligning residue $a$ with residue $b$, and the problem is to calculate a score for the conservation of the $j$ th position in the alignment. An $R$-dimensional space then is defined in which each column of the alignment can be considered. For a specified position $j$ in the alignment, each sequence consists of a single residue that is assigned a point $S$ in the space. For sequence $i$, position $j$, the point $S$ is defined as follows:

$$S = \begin{bmatrix} C(1, A_{ij}) \\ C(2, A_{ij}) \\ \ldots \\ C(R, A_{ij}) \end{bmatrix}$$

We then calculate a consensus value X for the $j$-th position in the alignment. X is defined as follows:

$$X = \begin{bmatrix} \dfrac{\sum\limits_{i=1}^{R} F_{ij} \times C(i, 1)}{M} \\ \dfrac{\sum\limits_{i=1}^{R} F_{ij} \times C(i, 2)}{M} \\ \ldots \\ \dfrac{\sum\limits_{i=1}^{R} F_{ij} \times C(i, R)}{M} \end{bmatrix}$$

where $F_{i,j}$ is the count of residues $i$ at position $j$ in the alignment.

Now, if $S$ is the position of sequence $i$ in the $R$-dimensional space, then we can calculate the distance $D_i$ between each sequence residue $i$ and the consensus position $X$.

$$D_i = \sqrt{\sum_{r=1}^{R} (X_r - S_r)^2}$$

where $X_r$ is the $r$ th dimension of position $X$, and $S_r$ is the r-th dimension of position $S$.

We define the quality score for the $j$-th position in the alignment as the mean of the sequence distances $Di$:

$$\text{Score} = \frac{\sum_{i=1}^{M} D_i}{M}$$

Finally, the scores are normalized by multiplying by the percentage of sequences that have residues (and not gaps) at this position.

***14.4.1.2 Exceptional Residues.*** Outlier residues are defined as those residues in the column conservation calculations described, which are found a long way from the consensus point (*i.e.*, which have a large distance $D_i$), thus, lowering the quality score for the column. For the $j$-th position in the alignment, only the sequences that have a residue at this position (and not a gap) are considered. We then calculate the upper and lower quartiles (the distances lying one-quarter of the way from the top and bottom of the array, respectively) and the interquartile range (the difference between the two quartiles) of the distances $D_i$ for this set of sequences. A residue $A_{ij}$ is considered an exception if the sequence distance $D_i$ is greater than (upper quartile + inter quartile range × scaling factor). The scaling factor can be adjusted by the user to select the proportion of residue exceptions that will be highlighted in the alignment display.

***14.4.1.3 Low-Scoring Segments.*** Given this alignment of $M$ sequences of length $N$ and a residue exchange matrix, we can build a profile that is weighted for sequence divergence. The weights are calculated directly from a neighbor joining tree, using the "branch-proportional" method described earlier, which corrects for unequal representation by down-weighting similar sequences and up-weighting divergent ones. Each sequence is assigned a weight $W_i$. In the residue comparison matrix $C$, the scores for common residue substitutions are positive, whereas rarer substitutions are scored negatively. The profile $P$ has a column of scores for each position in the alignment. The column is of height $R$ and consists of a score for each residue in the matrix $C$. The profile score for residue $r$ at position $j$ in the alignment is defined as follows:

$$P\,(r, j) = \frac{\sum_{i=1}^{M} C(r, A_{ij}) \times W_i}{\sum_{i=1}^{M} W_i}$$

For the $j$-th position in the $i$-th sequence the score $S_{ij}$ is defined as follows:

$$S_{ij} = P(A_{ij}, j)$$

The low-scoring regions in the $i$-th sequence are found by summing the scores $S_{ij}$ along the alignment in both the forward and the backward directions. The forward phase can be described by the following recurrence relations:

$$F_j = \begin{cases} F_{j-1} + S_{ij} & \text{if} \quad F_{j-1} + S_{ij} < 0 \\ 0 & \text{if} \quad F_{j-1} + S_{ij} \geq 0 \\ 0 & \text{if} \quad j = 0 \end{cases}$$

Having found the regions in the sequence that have negative $F_j$ scores, these regions then are refined by removing those positions at the end of each segment that have a positive profile score $S_{ij}$.

In a similar way, the backward phase can be described as follows:

$$B_j = \begin{cases} B_{j+1} + S_{ij} & \text{if} \quad B_{j+1} + S_{ij} < 0 \\ 0 & \text{if} \quad B_{j+1} + S_{ij} \geq 0 \\ 0 & \text{if} \quad j = N + 1 \end{cases}$$

The regions in the sequence that have negative $B_j$ scores again are refined by removing those positions at the beginning of each segment that have a positive profile score $S_{ij}$. The calculation is repeated for each sequence compared with a profile for all aligned sequences, except itself, and the low-scoring segments then are defined as those positions for which both $F_j$ and $B_j$ are negative.

## 14.5 CLUSTALW AND CLUSTALX 2.0

The developments described in the previous sections, mean that ClustalW and ClustalX can be used to produce high-quality, reliable multiple alignments for many real-world problems. Nevertheless, work to improve the software is ongoing, and many enhancements have been introduced since the original publication of the algorithms. For example, a faster implementation of the NJ algorithm now is used to construct guide trees during the multiple alignment process and also to construct phylogenetic trees based on the final alignment [5]. Major progress also has been achieved with the development of parallel versions of ClustalW and ClustalX by SGI and others, which show increased speeds and significantly reduce the time required for data analysis. The latest implementations of the software (versions 2.0) were described in a recent publication, by [14]. The programs were rewritten completely in C++ with a simple object model to make it easier to maintain the code and, more importantly, to make it easier to modify or even replace some of the alignment algorithms. For example, this version of Clustal offers both the UPGMA and the NJ methods for constructing the guide tree. Although the NJ tree is generally more accurate, the algorithm is time-consuming for very large datasets. UPGMA represents a more efficient option, which is more suitable for large-scale, high-throughput projects. The graphical interface in ClustalX 2.0 also was recoded using the portable Qt GUI toolbox.

A major new feature included in this version is the ability to improve alignment accuracy automatically using an iterative algorithm. Iteration is a quick and effective method of refining alignments. A “remove first” iteration scheme, which optimizes a weighted sum-of-pairs (WSP) score, has been included in this version of Clustal. During each iteration step, each sequence is removed from the alignment in turn and realigned. If the WSP score is reduced, then the resulting alignment is retained. The iteration scheme can be used either to refine the final alignment or at each step in the progressive alignment. Iterating during the progressive alignment is much more time consuming, as there are 2N-3 nodes in the guide tree, but it also tends to result in more accurate alignments.

## 14.6 DBCLUSTAL

Today, a new dimension is emerging, thanks to the systematic application of high-throughput genomics technologies and the resulting complete genomes, transcriptomes, proteomes, interactomes, and so on. This wealth of data provides unique opportunities to study complex biological systems, but it also clearly requires the development of algorithms and methodologies capable of handling the very large, complex datasets and of providing reliable, automatic analyses.

The complexity of today’s multiple alignment problem has incited several research teams to investigate combinations of different alignment algorithms and incorporation of biological information other than the sequence itself. For example, a comparison of several local and global protein alignment methods based on the BAliBASE benchmark [29] showed that no single algorithm was capable of constructing accurate alignments for all test cases. In particular, it was shown that global alignment methods were more accurate for the alignment of sequences that were homologous along their full lengths, but the local methods were more successful at identifying conserved regions when the sequences were only partially related. A similar observation was made in another study of RNA alignment programs [7], in which algorithms that incorporated structural information outperformed pure sequence-based methods for divergent sequences.

To address these problems, a modified version of Clustal, called DbClustal [30], was developed to allow the incorporation of local conservation information in the global alignment in the form of anchor points between pairs of sequences. An anchor point describes a locally conserved motif that is shared between two sequences. DbClustal reads an input file with a list of anchors in which each anchor is entered on a single line with the following format:

seq: NAME1 NAME2 beg: R1 R2 len: L weight: W

where NAME1 and NAME2 are the names of the two sequences, 1 and R2 are the first residues in the motif for sequences 1 and 2, respectively, L is the length of the anchor, and W is the weight or score of the anchor.

### 14.6.1 Anchored Global Alignment

In the multiple alignment stage of Clustal, two sequences with residues $a_1,\ldots, a_M$ and $\mathrm{b}_1,\ldots,\mathrm{b}_N$ are compared, and the optimal alignment is selected with the best sum-of-pairs score. The sum-of-pairs score is based on scores $S_{i,j}$ for aligning residues $a_i$ and $b_j$, and gap penalties for opening and extending a gap. In ClustalW, the score $S_{i,j}$ is simply equal to the residue comparison matrix score $C_{i,j}$ for the two residues. The alignment of two groups of sequences (or profiles) is a simple extension of the algorithm in which the score for aligning two residues is replaced by the score for aligning two columns in the respective profiles.

The score for aligning two residues (or profile columns) has been modified further in DbClustal to incorporate local anchors. During the progressive multiple alignment, an $M \times N$ position-specific anchor matrix is calculated for each pair (or group) of sequences to be aligned. For column $i$ in the first group of sequences and column j in the second group, the anchor matrix score $\mathrm{Anchor}_{ij}$ is:

$$\mathrm{Anchor}_{i,j} = \mathrm{Max}(0, W_k)$$

for all anchors containing any pair of residues in columns $i,j$, where $W_k$ is the weight defined in the anchor input file.

For a pair of sequences, the score for aligning residues $A_i$ and $B_j$ is then defined as follows:

$$C_{i,j} + \mathrm{Anchor}_{i,j}$$

where $C_{i,j}$ is the residue comparison matrix score for $A_i$ and $B_j$.

Similarly, the score for aligning two groups of sequences is defined as follows:

$$P_{i,j} + \mathrm{Anchor}_{i,j}$$

where $P_{i,j}$ is the profile-to-profile score for $A_i$ and $B_j$.

The penalties for opening and extending gaps remain the same.

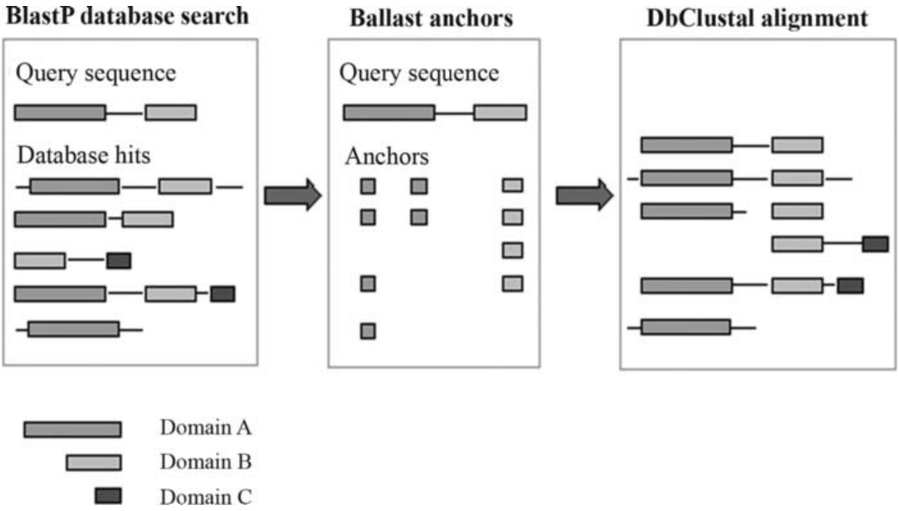


**Figure 14.8** Schematic representation of the integration of Ballast anchors in DbClustal. A theoretical query sequence is shown possessing two functional domains. This query is used to perform a database search with the BlastP program. BlastP detects sequences sharing at least one domain with the query sequence. The BlastP results then are processed by Ballast to identify locally conserved motifs. Finally, the motifs are used as anchors for the DbClustal global alignment.

The anchor points can be constructed from any local alignment algorithm, such as Dialign [18] or the Ballast program [23]. Figure 14.8 illustrates this approach with conserved motifs extracted from the top sequences detected by a BlastP database search [2] using the Ballast program.

By combining the advantages of both global and local alignment methods into a single system, DbClustal represents a significant step toward the automatic construction of high-quality alignments of large families of complex sequences, for example, with large N/C-terminal extensions or internal insertions, or multiple structural or functional domains.

## 14.7 PERSPECTIVES

Multiple alignment methods currently are evolving away from a single isolated algorithm toward more cooperative systems based on the exploitation of additional information (3-D structures, function, and evolution), and the development of the Clustal family of alignment programs clearly will follow this trend. In this context, the recent redesign of the code in an object-oriented language was aimed at facilitating the integration of novel algorithms in the field.

The potential of this approach was estimated in a recent study to test and evaluate several more successful algorithms (Figure 14.9), which demonstrated that the execution time and the quality of the resulting multiple alignments could be improved

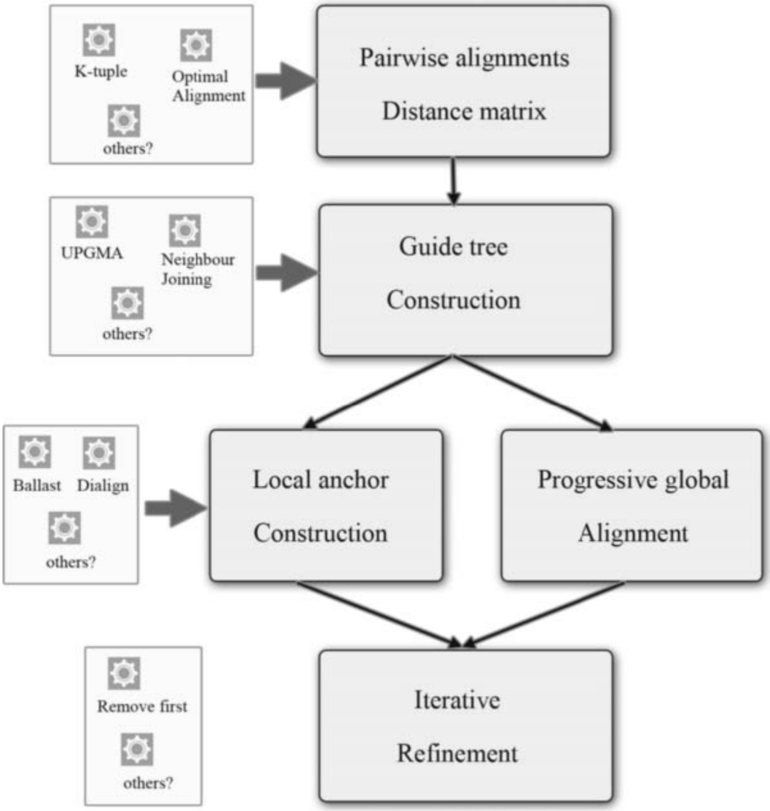


**Figure 14.9** Evolution of the Clustal programs.
Creation and integration of alternative modules at each alignment stage: (A) pairwise distance calculation, (B) guide tree construction, (C) detection of local "anchors," and (D) iterative refinement.

significantly [3]. Future versions of Clustal undoubtedly will incorporate some, or all, of these complementary algorithm, as well as other data mining composants, statistical analyses, and so on. The developments will allow us to create an integrated system to test, evaluate, and optimize each step in the construction and subsequent analysis of a multiple alignment.

## REFERENCES

1. S.F Altschul and B.W Erickson. Optimal sequence alignment using affine gap costs. *Bull Math Biol*, 48:603–616, 1986.
2. S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–3402, 1997.

3. M.R. Aniba, S. Siguenza, A. Friedrich, F. Plewniak, O. Poch, A. Marchler-Bauer, and J.D. Thompson. Knowledge-based expert systems and a proof-of-concept case study for multiple sequence alignment construction and analysis. *Brief Bioinformatics*, 10:11–23, 2009.
4. S.A. Benner, M.A. Cohen, and G.H. Gonnet. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. *J Mol Biol*, 229:1065–82, 1993.
5. R. Chenna, H. Sugawara, T. Koike, R. Lopez, T.J. Gibson, D.G. Higgins, and J.D. Thompson. Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Res*, 31, 3497–3500, 2003.
6. D.F. Feng and R.F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol*, 25, 351–360, 1987.
7. P.P. Gardner, A. Wilm, and S. Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Res*, 33:2433–9, 2005.
8. O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162:705–8, 1982.
9. M. Gribskov, A.D. McLachlan, and D. Eisenberg. Profile analysis: detection of distantly related proteins. *Proc Natl Acad Sci USA*, 84:4355–8, 1987.
10. S. Henikoff and J.G. Henikoff. Performance evaluation of amino acid substitution matrices. *Proteins*, 17:49–61, 1993.
11. D.G. Higgins and P.M Sharp. CLUSTAL: a package for performing multiple sequence alignments on a microcomputer. *Gene*, 73:237–244, 1988.
12. D.G. Higgins and P.M Sharp. Fast and sensitive multiple sequence alignments on a microcomputer. *CABIOS*, 5:151–153, 1989.
13. D.G. Higgins, A.J. Bleasby, and R. Fuchs. CLUSTAL V: improved software for multiple sequence alignment. *Comput Appl Biosci*, 8:189–191, 1992.
14. M.A. Larkin, G. Blackshields, N.P. Brown, R. Chenna, P.A. McGettigan, H. McWilliam, F. Valentin, I.M. Wallace, A. Wilm, R. Lopez, J.D. Thompson, T.J. Gibson, and D.G. Higgins. Clustal W and Clustal X version 2.0. *Bioinformatics*, 23:2947–2948, 2007.
15. O. Lecompte, J.D. Thompson, F. Plewniak, J. Thierry, and O. Poch. Multiple alignment of complete sequences (MACS) in the post-genomic era. *Gene*, 270:17–30, 2001.
16. R. Luthy, A.D. McLachlan, and D. Eisenberg. Secondary structure-based profiles: use of structure-conserving scoring tables in searching protein sequence databases for structural similarities. *Proteins*, 10:229–39, 1991.
17. R. Luthy, I. Xenarios, and P. Bucher. Improving the sensitivity of the sequence profile method. *Protein Sci*, 3:139-46, 1994.
18. B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–4, 1998.
19. E.W. Myers and W. Miller. Optimal alignments in linear space. *Comput Appl Biosci*, 4:11–17, 1988.
20. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48:443–453, 1970.
21. P.C. Ng, J.G. Henikoff, and S. Henikoff. PHAT: a transmembrane-specific substitution matrix. *Bioinformatics*, 16:760–766, 2000.
22. S. Pascarella and P. Argos. Analysis of insertions/deletions in protein structures. *J Mol Biol*, 224:461–71, 1992.

23. F. Plewniak, J.D. Thompson, and O. Poch. Ballast: blast post-processing based on locally conserved segments. *Bioinformatics*, 16:750–9, 2000.

24. N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Evol Biol*, 4:406–425, 1987.

25. P.H. Sneath and R.R. Sokal. In “Numerical taxonomy.” W.H. Freeman, San Francisco, CA, 1973.

26. W.R. Taylor. A flexible method to align large numbers of biological sequences. *J Mol Evol*, 28:161–169, 1988.

27. J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22:4673–4680, 1994.

28. J.D. Thompson, T.J. Gibson, F. Plewniak, F. Jeanmougin, and D.G. Higgins. The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Res*, 25:4876–4882, 1997.

29. J.D. Thompson, F. Plewniak, and O. Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15:87–8, 1999.

30. J.D Thompson, F. Plewniak, J.C Thierry, and O. Poch. DbClustal: rapid and reliable global multiple alignments of protein sequences detected by database searches. *Nucleic Acids Res*, 28:2919–2926, 2000.

31. M. Vingron and P.R. Sibbald. Weighting in sequence space: a comparison of methods in terms of generalized sequences. *Proc Natl Acad Sci USA*, 90:8777–81, 1993.

32. C.R. Woese and N.R. Pace. Probing RNA structure, function and history by comparative analysis. “The RNA World.” Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY, 1993.

33. W.J. Wilbur and D.J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci USA*, 3:726–30, 1983.

34. M.O. Dayhoff, R.M. Schwartz, B.C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*. National Biomedical Research Foundation: 345–352, 1978.

# 15

# FILTERS AND SEEDS APPROACHES FOR FAST HOMOLOGY SEARCHES IN LARGE DATASETS

Nadia Pisanti, Mathieu Giraud, and Pierre Peterlongo

## 15.1 INTRODUCTION

### 15.1.1 Homologies and Large Datasets

Homologies inside large sequences or a large set of sequences are the key to several molecular biology studies. Similarities between genomic sequences are often traces of common ancestry, and the study of distances between species teaches us about the history of the evolution. Conserved elements between distant species are genes, transcription factors binding sites, transposable elements, or other functional elements.

Basically, homology-finding algorithms aim to detect in nucleic sequences more or less similar fragments, called simply *repeats*. Such fragments can be found within one sequence or in a set of several sequences. The selection pressure is not focused on the only nucleic sequences; for proteins, comparisons on the proteic sequences are often more relevant, and for RNA, the secondary structure can be more conserved than the nucleic sequence [14]. Similarities between sequences are often a first step to other more specific tools applied to the study of particular conserved elements.

On the other hand, the amount of data that biologists are dealing with are growing exponentially. Another recent reason for this relies on next generation

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

sequencers [56]; they enable faster sequencing of DNA and with lower costs (several orders of magnitude cheaper) than using the original Sanger et al. [54] method. These recent sequencers open the way to new horizons in molecular biology. As claimed in Mardis' review [40], "an astounding potential exists for these technologies to bring enormous change in genetic and biological research and to enhance our fundamental biological knowledge." Extracting pertinent knowledge from these floods of data requires to find similarities and patterns quickly and efficiently.

The exhaustive similarity search, implying dynamic programming [58], suffers from a time bottleneck. If no heuristic is used, then finding similarities between sequences requires a time proportional to the product of their lengths. Such approaches are difficult with large datasets. In practice, the exhaustive similarity search is reserved for small datasets and when the application requires very precise results. On larger datasets, similarities are found using preprocessing or heuristics through *filters*. Some filters have huge success; the most used tool in bioinformatics for sequences comparison in the last two decades, BLAST [2, 1], is based on a filtering heuristics.

### 15.1.2 Filter Preprocessing or Heuristics

Filters approaches are based on the following idea: occurrences of an approximately repeated fragment must share a certain number of short fragments that are exactly conserved. The search of repeats thus shall focus on regions of high enough concentration of these shared fragments. This idea can be used either for preprocessing data (removing as much as possible portions that can not contain occurrences of repeats), or as a heuristic (anchoring the search of the repeats using the so-called seeds).

In both cases, these approaches are powerful for quickly finding similarities with full sensibility (without missing any information) or high sensibility (possibly missing some information). Since the early 1990s, seed-based filters have been developed in two different directions. *Lossless filters* guarantee that no occurrence is missed (typically exhibiting a seed-based condition that is proven to be necessary and that it is easy to check), whereas *lossy seed filters* propose some seed models as a starting point (and then explore the properties of those models to assess the sensitivity performances of the filter).

Actually, lossless filters and lossy seed filters are highly related. The fundamental idea of both approaches is to focus directly on sequences fragments that are likely to provide a similarity, getting rid of useless computations that may be avoided. There is no precise border between these two concepts.

### 15.1.3 Contents

Within various contexts, the authors of this chapter developed methods based on lossless filters or on seed-based lossy filters. This chapter will present the lossless filters and the seed-based approaches by describing a brief state-of-the-art process for each method and by presenting the most recent works in these fields. The following section gives some common definitions and introduces basic concepts.

## 15.2 METHODS FRAMEWORK

### 15.2.1 Strings and Repeats

We introduce here the terminology used in the forthcoming sections. A *string* is a concatenation of zero or more symbols from an alphabet $\Sigma$. A string $s$ of length $n$ on $\Sigma$ is represented also by $s[0]s[1]\dots s[n-1]$, where $s[i] \in \Sigma$ for $0 \leq i < n$. The length of a string $s$ is denoted by $|s|$. We denote $s[i, j]$ the *substring* $s[i]s[i+1]\dots s[j]$ of $s$. In the following, we also use the notion of *$q$-gram.* A $q$-gram is a word (a short string) of length $q$. If a $q$-gram occurs in two strings $\omega$ and $\omega'$, then this $q$-gram is said to be *shared* by $\omega$ and $\omega'$.

We recall that the *Hamming distance* between two words of the same length is the minimal number of substitutions needed to transform the first into the other, whereas the *edit distance* between two words (not necessarily of the same length) is the minimum number of substitutions, insertions and deletions to transform the first into the other. We denote by $d_H(\omega, \omega')$ (respectively, $d_E(\omega, \omega')$) the Hamming distance (respectively, the edit distance) between the two strings $\omega$ and $\omega'$.

A set of words whose pairwise Hamming distance or edit distance is bounded by a given threshold is called an *approximate repeat.* To simplify the reading, in the following, the term "repeat" will design an approximate repeat. A *multiple repeat* is a repeat with at least three words. We focus on $(L, r, d)$*-Hrepeat* and $(L, r, d)$*-Erepeat*, defined as follows:

**Definitio 15.1 ($(L, r, d)$- Hrepeat)** *Given a set $\mathcal{S}$ of one or more input strings, a length $L > 0$, an integer $r \geq 2$, and a Hamming distance $0 \leq d < L$, we call a $(L, r, d)$-Hrepeat a set $\{\omega_1, \dots, \omega_r\}$ of $r$ words of length $L$ occurring in the sequences of $\mathcal{S}$ such that for all $i, j \in [1, r]$, $d_H(\omega_i, \omega_j) \leq d$.*

**Definitio 15.2 ($(L, r, d)$-Erepeat)** *Given a set $\mathcal{S}$ of one or more input strings, a length $L > 0$, an integer $r \geq 2$, and an edition distance $0 \leq d < L$, we call a $(L, r, d)$-Erepeat a set $\{\omega_1, \dots, \omega_r\}$ of $r$ words having a length of at least $L - d$ occurring in the sequences of $\mathcal{S}$ such that for all $i, j \in [1, r]$, $d_E(\omega_i, \omega_j) \leq d$.*

Both definitions can be used to study repeats inside one sequence ($|\mathcal{S}| = 1$) or between several sequences ($|\mathcal{S}| > 1$). In the latter case, one also can enforce that the $r$ words occur over $r$ distinct sequences (and thus one needs $|\mathcal{S}| \geq r$).

### 15.2.2 Filters—Fundamental Concepts

For finding multiple repeats, exhaustive methods based on dynamic programming suffer from a theoretical time complexity in $O(n^r)$, where $n$ is the size of each sequence and $r$ is the number of sequences (or the number of repeats searched in a unique sequence). Some optimizations based on string compression achieve a sub-$O(n^r)$ complexity [10], but applications on large datasets remain very difficult. The goal of filters thus is to reduce this factor $n$ considerably by getting rid of almost all the useless data that only would slow down the real search. Repeats then could be sought in a much reduced dataset. Ideally, this dataset would contain only the searched repeats. All filters start from the same observation that can be explained with the simple Example 15.1: two similar words share at least a certain number of $q$-grams.

### ■ EXAMPLE 15.1 Shared *q*-Gram

Words ATAGGAT and ATATGAT are two words of length 7 with Hamming distance equal to 1 (one substitution position 4). Occurring in a set of sequences $\mathcal{S}$, they are occurrences of a (7,2,1)-Hrepeat. These two words share the 3-gram ATA at position 0 and the 3-gram GAT at position 4.

Filters thus are meant as a preprocessing task to any algorithm that finds and localizes repeats, and hence, they can be employed as a preliminary step to any tool designed for finding repeats or an application using repeats.

With *lossless filters*, we refer to methods that filter the data ensuring that no fragments that may contain a repeat are removed; there are no false negatives. In general, however, there can be false positives; otherwise, there would be no difference between a filter (required to be fast) and an exhaustive search (inevitably slow). Besides the speed requirement, a filter is powerful if it is *selective*; that is, it leaves the least amount of false positives.

On the other hand, *lossy filters* may produce false negatives; preprocessing data with such filters may modify the final result. A good lossy filter must be as selective as possible but also as *sensitive* as possible; that is, it generates the least amount of false negatives, thus approaching the full sensitivity guaranteed by lossless filters.

Being lossless or lossy depends on the design of filter, but the same filter can be used for distinct *repeat models* (the kind of repeats searched, their required length, and their minimal frequency) and, hence, switching from lossless to lossy if the conditions happen to require more speed over precision. For example, as shown in Example 15.1, requiring to find at least two (possibly overlapping) 3-grams between words of length 7 leads to a lossless filter for (7,2,1)-Hrepeats. This means that the two members of any (7,2,1)-Hrepeat share at least two 3-grams. However, if the same condition (at least two 3-grams shared) is used for filtering for (7,2,2)-Hrepeats, then the filter become lossy. Its sensitivity can be measured; only 28.5% couples of words $\{\omega, \omega'\}$ with $|\omega| = |\omega'| = 7$ and $d_H(\omega, \omega') \leq 2$ share at least two 3-grams. The filter has thus a 28.5% sensibility.

Moreover, the computation of sensitivity and specificity also depends on the *background probability model*. The 28.5% sensibility of the previous filter was computed on a Bernoulli model consisting of independent and identically distributed nucleotides. Other more elaborated models better reflect the exact nature of biological sequences.

Although some filters can be used as a generic preprocessing step to any tools that finds repeats, specific filters thus are designed often for a particular repeat model and probability model.

**Table 15.1 Methods discussed in this chapter**

| | Hamming distance<br>(substitutions) | Edit distance<br>(+ indels) |
|---|---|---|
| Lossless (sensitivity $= 1$) | NIMBUS | QUASAR, SWIFT, TUIUIU |
| Lossy (sensitivity $\leq 1$) | spaced seeds, and their optimizations | |

In the following, we expose some recent filtering methods as summed up in Table 15.1. Section 15.3 details lossless filters, and Section 15.4 details lossy seed filters.

## 15.3 LOSSLESS FILTERS

For large or complex problem instances, lossless filters help to get exact solutions that would have been otherwise prohibitive. They also speed up other heuristics. This section presents a brief history of lossless filters as well as a description of four of these filters. In all cases, the rationale first is to *detect and prove a necessary condition* for a fragment to be part of a repeat and then to *design a fast method* that checks this condition. To be as efficient as possible, the necessary condition should be designed *ad hoc* for the kind of sought repeat. For example, it does matter whether insertions and deletions are admitted; filters on Hamming or edit distance basically only share the rationale we just described.

### 15.3.1 History of Lossless Filters

Already in 1987, filtering has been suggested as a screening task to speed exact pattern matching algorithms; in [21], an efficient hash function was suggested for these purposes. The first screening method explicitly designed for finding repeats was suggested in [22] as an online algorithm that searches for "frequent elements" in stream data; also in the latter case, the elements to be searched were exact, that is, their occurrences all are required to be identical. The first time that a screening was devised that took into account approximation was in [17] where up to a certain number of mismatches are allowed (a bounded Hamming Distance is tolerated) but only with the purpose of finding (approximate) occurrences of a given pattern inside a given text.

Specifically aiming at computational biology applications, filtering has been employed successfully by several tools as a preprocessing to approximate pattern matching tasks. The QUASAR (Section 15.3.2, [7]) method first applies a necessary condition for two strings to be similar then finds all matches that admit a given limited number of edit operations. This necessary condition also has been used in SWIFT [52] to design a filter that is an evolution of QUASAR. The SWIFT algorithm adds the use of parallelograms in the necessary conditions (see Section 15.3.2). This approach leads to faster and more selective tools.

Both QUASAR and SWIFT filters search for fragments of the text that fulfill a condition that can be seen as a requirement for two words to belong to a $(L, 2, d)$-Erepeats. In this sense, they are both ancestors of NIMBUS (sections 15.3.3) and TUIUIU (Section 15.3.4), filters for finding *multiple* repeats.

### 15.3.2 QUASAR and SWIFT—Filtering Repeats with Edit Distance

The necessary condition of QUASAR [7] and SWIFT [52] tools can be embedded in any tool, exact or heuristic, that preprocesses the search of long repeats that consist of multiple occurrences with a limited number of insertions, deletions, and substitutions. By long repeats, we refer to repeats whose length of each occurrence is about 50 nucleotides and more.

With respect to QUASAR, SWIFT presents some improvements that lead to shorter execution time and to more specific filtering.

***15.3.2.1 Necessary Condition.*** QUASAR and SWIFT apply the following condition to filter for $(L, 2, d)$-Erepeat. This condition was first introduced in [60].

**Theorem 15.1** *The minimum number of $q$-grams that words of an $(L, 2, d)$-Erepeat must share is*

$$p_2 = L - q + 1 - qd$$

Thus, while filtering for finding $(L, 2, d)$-Erepeats, all sequence fragments of length $L$ that do not share at least $p_2$ $q$-grams with another fragment are filtered out. Note that $q$, user-defined, is thus one of the main parameter of such approaches.

***15.3.2.2 QUASAR Implementation.*** The QUASAR tool slides a window $w$ of length $L$, checking whether in the sequence there is another fragment of length at most $L - d$ that shares at least $p_2$ $q$-grams with $w$. For finding sets of shared $q$-grams, the sequences are partitioned into blocks of size $b \geq 2L$ occurring at every $b$ position. Each such block overlaps by at least $L$ characters with its predecessor. Such an approach ensures that any occurrence of a word of an $(L, 2, d)$-Erepeat is always totally contained in at least one such block. The $q$-grams of each block are indexed using a suffix array. The sliding window is retained if it shares at least $p_2$ $q$-grams with at least one such block.

***15.3.2.3 SWIFT Implementation.*** SWIFT is an evolution of QUASAR. The main improvement relies on the use of restricted *parallelograms* that are a shaped area that limits the space search of the shared $q$-grams. Figure 15.1 shows an example of such a parallelogram. Should two strings $w$ and $w'$ have an edit distance no greater than $d$, then in the dynamic programming matrix used for computing their alignment within the edit distance, there would be an optimal alignment consisting in a path making at most $d$ vertical or horizontal steps and, thus, involving at most $d$ consecutive

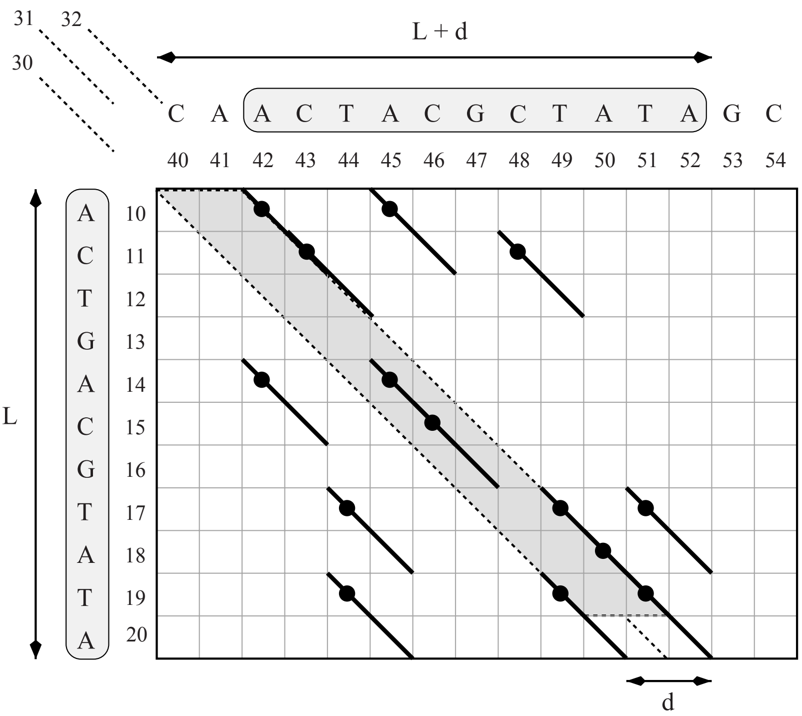


**Figure 15.1** An example a (11, 2, 2)-repeat found using 2-grams. The shown parallelogram defines portions of sequences in which shared 2-grams must be searched for filtering for finding (11, 2, 2)-repeats.

diagonals. Because the matches caused by the mandatory $q$-grams shared by $w$ and $w'$ necessarily will belong to this path, they would be forced to be in an area restricted by two diagonals that are $d$ positions apart—a parallelogram.

In practice, searching shared $q$-grams in the area restricted to the parallelograms is an elegant approach leading to major improvements. First, it enables increasing the speed of computations by limiting the working space. Second, it increases the specificity of the filter, avoiding false positives because of $q$-grams outside the parallelograms.

### 15.3.3 NIMBUS—Filtering Multiple Repeats with Hamming Distance

The NIMBUS tool is based on the Hamming distance. It has been the first filter designed directly for filtering while searching *multiple* repeats instead of repeats having only two occurrences. The tool NIMBUS [49, 50] can be employed as a preprocessing step to *any* tool that searches for long multiple repeats with mismatches or that performs alignments based on the detection of such local repeats. Being a lossless filter, this tool can be used as a preliminary step of both heuristics and exact methods.

#### *15.3.3.1* NIMBUS *Necessary Condition.*

NIMBUS uses the following condition to filter for $(L, r, d)$-Hrepeats:

**Theorem 15.2** *The minimum number of nonoverlapping $q$-grams that words of an $(L, r, d)$-Hrepeat must share is*

$$p_r = \left\lfloor \frac{L}{q} \right\rfloor - d - (r-2) \times \left\lfloor \frac{d}{2} \right\rfloor$$

NIMBUS uses Theorem 15.2 and hence detects and removes all sequences fragments of length in $[L-d, L+d]$ that do not share at least $p_r$ $q$-grams with at least $r-1$ other fragments of length in $[L-d, L+d]$. For $r = 2$, the formula of Theorem 15.2 coincides with Theorem 15.1.

We now give an insight of the proof. The full proof can be found in [49]. Let us consider the hypothetical alignment of $r$ words of length $L$ of a $(L, r, d)$-Hrepeat. Suppose $d = 0$ (words are all identical), then they share exactly $\left\lfloor \frac{L}{q} \right\rfloor$ nonoverlapping $q$-grams. Now, if $d > 0$, then the number of shared $q$-grams obviously decreases. In the worst case scenario, every pair of the $r$ strings has Hamming distance $d$. For each position $i$ in which there is a letter substitution between any pair of strings (positions represented by an x in Figure 15.2), no shared $q$-gram can include that position, meaning that up to $q$ of them are excluded. Given that there are a total of $d \times r(r-1)/2$ positions in which there is a mismatch, at worst $q \times d \times r(r-1)/2$ $q$-grams are excluded, and thus, $p'_r = \left\lfloor \frac{L}{q} \right\rfloor - (q \times d \times r(r-1)/2)$ shared $q$-grams must be left. This would be a very weak necessary condition for a filter.

Observing that the positions of the mismatches between pairs of strings must necessarily overlap, the stronger bound of Theorem 15.2 is obtained. To see why the positions of mismatches must overlap, let us consider the simple case of $r = 3$ strings $s_1, s_2$, and $s_3$, and $d = 1$. Let $a$ be the position of the unique mismatch between $s_1$ and $s_2$, let $b$ be that of the mismatch between $s_2$ and $s_3$, and let $c$ be that between $s_1$ and $s_3$. Because $s_1[a] \neq s_2[a]$, then $s_3[a]$ cannot match with both, and thus, either $s_3$ mismatches with $s_1$ at position $a$ (and hence $a = c$), or it mismatches with $s_2$ there, and therefore, $a = b$; summing up, column $a$ either coincides with $b$ or with $c$. Moreover (and symmetrically), at position $c$, it must be that $s_1$ has a mismatch

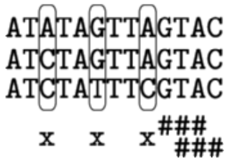


**Figure 15.2** Words belonging to a (13, 3, 2)-Hrepeat share at least $p_3$ nonoverlapping 3-grams with $p_3 = \left\lfloor \frac{13}{3} \right\rfloor - 2 - (3-2) \times \left\lfloor \frac{2}{2} \right\rfloor = 1$. Indeed, a 3-gram GTA, for instance, is shared by the three sequences. In this example, mismatches were spread every three characters. This repartition is the most limiting case for the number of shared 3-grams.

with $s_2$ or with $s_3$, and hence, actually when $d = 1$, it must be that $a = b = c$; there is a unique column that no $q$-gram can cross. Clearly, in the general case of $r > 3$ and $d > 1$, there are more than one position of mismatches, but the principles of the mandatory concentration of mismatches positions into columns can be generalized. Roughly speaking, for any $d$ and $r > 2$, each new word contributes by around $d/2$ new columns containing a mismatch.

***15.3.3.2 Multiple Sequences Versus a Single Sequence.*** It is worth noticing that Theorem 15.2 does not depend on the repartition of repeat occurrences over sequences. Thus, it applies either in the case of searching repeats occurring in a single sequence or distributed over at least $r$ sequences.

***15.3.3.3*** **Nimbus** ***Implementation: Bifactor Array.*** Sets of shared $q$-grams are searched efficiently thanks to *bifactors*. Given a sequence, a bifactor is simply two substrings separated with a gap. Example 15.2 shows a bifactor. Each couple of shared $q$-gram is a shared bifactor. The set of $p_r > 1$ shared $q$-grams are detected by finding a couple of shared $q$-grams $A - B$ as shared bifactors. Then a second shared bifactor $B - C$ (starting by $B$) is searched. In case of success, the group of three shared $q$-grams $A - B - C$ is found and so on. To speed the detection of shared bifactors, they are indexed in a specialized data structure called the *bifactor array* described in [50].

■ **EXAMPLE 15.2 Bifactor**

On the sequence $\text{T}\underline{\text{ATA}}\text{TA}\underline{\text{GTA}}\text{C}$, at position 1, occurs the bifactor ATA GTA, with two substrings of length three separated by a gap of length two. Bifactors are similar to spaced seeds that will be discussed in Section 15.4.2.

***15.3.3.4*** **Quasar** ***and*** **Nimbus** ***Implementations Differences.*** The algorithm of NIMBUS deals with repeats having possibly more than two occurrences, whereas QUASAR was designed for filtering for repeats having only two occurrences. Moreover, QUASAR does not check the order of the shared $q$-grams. As the edit and Hamming distances do not allow inversions, the shared $q$-grams must have the same repartition in each word of $(L, r, d)$-Hrepeats. Thus, the NIMBUS tool, in addition to allowing the finding of repeats with more than two occurrences, applies in practice a method with higher specificity.

***15.3.3.5 Performance.*** Preprocessing with NIMBUS a dataset in which one wants to find functional elements using a multiple local alignment tool such as Glam [16], the overall execution time could be reduced from 7.5 hours (directly aligning with GLAM only) to less than two minutes (filtering with NIMBUS and then aligning with GLAM) [49].

### 15.3.4 TUIUIU—Filtering Multiple Repeats with Edit Distance

The TUIUIU [51] tool proposes to extend the filtration for $(L, r, d)$-Erepeat with $r \geq 2$. The approach uses a filtration condition framework based on the $p_2$ number of shared $q$-grams (Theorem 15.1).

With $r \geq 2$, a necessary condition involving the number of $q$-grams that must be shared by *all* $r$ occurrences of the repeats would result as too weak here because, when indels are allowed, the property of mismatch columns that concentrate does not hold anymore. Therefore, the choice made in TUIUIU actually was to design a very strong necessary condition for two strings to be at most edit distance $d$ and to insert this checking in a suitable framework that detects fragments of the input data that fulfill the requirement with respect to at least $r - 1$ other fragments belonging to distinct input strings. Therefore, the contribution of the algorithm introduced in TUIUIU is twofold; first, a new necessary condition for $(L, 2, d)$-Erepeat is introduced, which results in being stronger that previous ones; second, the framework that extends the necessary condition to multiple repeats, which actually can be employed with any $(L, 2, d)$-Erepeat condition inside.

The necessary condition checked by TUIUIU is actually a series of three possible levels of selectivity, resulting in as many versions of the filter. The first condition (already introduced in [52]) requires that Theorem 15.2 holds with $r = 2$, and also that in the alignment matrix of the two strings, these $p_2 = \lfloor L/q \rfloor - d$ $q$-grams result in matches that lay in a parallelogram-shaped area (see Figure 15.1). The second (further) condition that TUIUIU imposes is very simple; for $w$ and $w'$ to be a $(L, 2, d)$-Erepeat, the $q$-grams that they must share have to occur in $w$ at distinct positions (that is, at least $p_2$ of them). This apparently trivial condition actually resulted in giving a substantial contribution to the strength of the filter in that TUIUIU can check it in negligible constant time, and it does increase the selectivity. For this reason, the performances of the filter version that use this condition clearly outperform those of the filter that uses the first condition only. The third and most stringent condition additionally imposes that there is a set of $p_2$ shared $q$-grams that occur in $w$ and $w'$ in the same order. This third condition, involving longest common subsequence (LCS) computations, checks the conservation in the order of the shared $q$-grams. It requires some extra time to be checked, but experiments showed evidence of the fact that because this is done only for pairs of strings $w$ and $w'$ that already survived the previous conditions, the delay is limited. In practice, this most restrictive constraints resulted in being worth using in many interesting applications, as, for example, while using values of $d$ larger than 10% of $L$.

The first is the fastest and less sensible and is, in practice, the same as that of SWIFT (the use of a parallelogram), except that it is used within the multiple repeats filtering task. The second option introduces an extra requirement that leads to a more selective tool while being still as fast as before (or actually sometimes even faster). This condition actually could be itself a good filter for finding approximate matches of a pattern into a text. The third choice results in the most selective filter but at a time cost that experiments show to be often worth paying when the target task is finding multiple repeats.

To require that the repeat occurs in $r$ distinct sequences, TUIUIU slides a window over all input sequences. At each moment, it considers the window itself $w$ and all remaining sequences virtually divided into blocks that are candidate to contain $w'$. For the first position of the window, it builds an index of all its $q$-grams and stores how many of them belong to each block. For every new position of the window, updating this information is very simple as $w$ simply drops a $q$-gram and acquires a new one. It is thus also easy to check, for each block, whether it has enough shared $q$-grams. If for $w$, there are enough blocks that satisfy the retained, then $w$ is conserved; otherwise, $w$ is filtered out.

Like NIMBUS, TUIUIU also supports a query in which there is a single input sequence, and the $r$ occurrences of the repeat only are required to be distinct as they all must belong to the same sequence.

***15.3.4.1 Complexity and Parameters.*** In general (with $p_2$ *large enough*, see [51]), the average complexity is in

$$O\left(\frac{b+d}{b}n^2|\Sigma|^{-q}\right)$$

where $n$ is the sum of the sequence lengths and $b$ the thickness of parallelograms. It is worth noting that $q$, the length of the $q$-grams, strongly influences the results and the computation time. With a small $q$, computation is slow (a lot of shared randomly $q$-grams), but the filter is very specific. On the other hand, a large value for $q$ increases the theoretical speed both because of the $|\Sigma|^{-q}$ term in the complexity but also because only few large $q$-grams are shared randomly between sequences. However, with a larger $q$ value, the filter becomes less selective. This is a result of the decrease of the strength of the necessary condition on $p_2$. In practice, as presented in [51], applying TUIUIU on biological sequences with $q = 6$ presents a good balance between specificity and execution time.

***15.3.4.2 Success Story.*** TUIUIU was applied as a preprocessing step of a multiple alignment application (GLAM2 [15], an evolution of GLAM that allows indels and is thus more suitable with edit distance), leading to an overall execution time (filter plus alignment) on average of 63 and at best 530 times smaller than before (direct alignment).

## 15.4 LOSSY SEED-BASED FILTERS

When dealing with weaker similarities, lossless filters become too difficult to be implemented efficiently. As an example, when we search for (40, 2, 10)−Hrepeats (that means a $\geq$ 75% similarity), Theorem 15.2 cannot ensure anything for $q$-grams with $q \geq 4$. In some homology studies between "distant" species, one frequently has to look at similarity levels between 30% and 60%; for those kind of similarities, tools

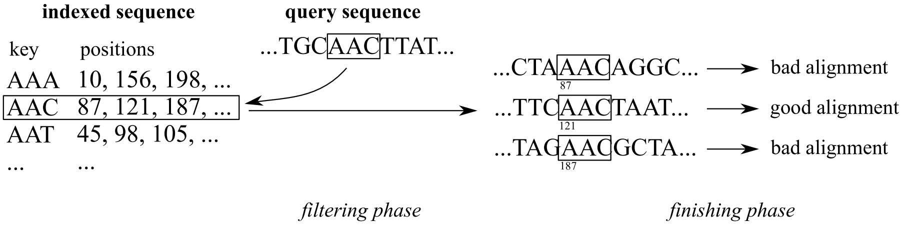


**Figure 15.3** During the filtering phase, a scan of the query sequence uses the index (here with the key AAC) to identify all matching seeds. Here the first $r = 3$ seeds are shown. Then the finishing phase get neighborhoods of the indexed sequence from the memory (one memory access per position) and builds local alignments between this neighborhood and the one of the query sequence. Here only the position 121 in the indexed sequence leads to a "good" local alignment.

other than lossless filters must be designed. In this section, we explain seed-based heuristics and detail some of their implementations.

### 15.4.1 Seed-Based Heuristics

As in lossless filters, the idea of seed-based heuristics is to anchor the detection of similarities using matching short words or short subsequences occurring in both compared sequences. The form of these words or subsequences is provided by a pattern called a *seed*. A word that respects the seed is called a *key*. For instance, MVK is one of the $20^3$ possible keys for the seed of three consecutive characters on the protein alphabet.

We now summarize the discussion made by the excellent survey of Brown ([5], page 122). Seed-based alignment proceeds in three phases (Figure 15.3):

- *Indexing phase:* "indexing one or more of the sequences"—typically in $O(n)$ time, or at most $O(kn)$ time, where $n$ is the size of the indexed sequence and $k$ is the size of keys. For each key, all positions of the occurrences in the database are stored.
- *Filtering phase:* "using the index to identify possible alignment seeds"—typically, each position in the query is processed in $O(1 + r)$ time, where $r$ is the number of matching seeds. Thus, this phase takes $O(m + R)$ time, where $m$ is the size of the query and $R$ is the total number of matching seeds.
- *Finishing phase:* "building local alignments from the seeds"—$O(R)$, assuming that most seeds are bad seeds (that are not extended into good alignments). To be efficient, the finishing step usually begins by a fast alignment between the query and the database on a small *neighborhood* around the seed match, then proceeds to a full alignment if the fast alignment was above a given threshold.

In fact, if we leave out the constant terms and the $O(m + n)$ time required to read the sequences, a seed-based alignment algorithm runs in additional $O(R)$ time;

the specificity of the filtering phase is crucial. Traditional BLAST seeds, noted as ###########, are contiguous 11-grams on nucleotides. Assuming the DNA sequences are random, $R$ is approximately $mn/4^{11}$. With large genomes, this quantity becomes too large.

### 15.4.2 Advanced Seeds

For the same number of detected good alignments (sensibility), how can it be possible to have less hits and thus to decrease $R$ ? Instead of contiguous $k$-words, it is more advantageous to use so-called *spaced seeds* that correspond to matches "with gaps" between sequences and thus gapped diagonals in the dynamic programming matrix. Hits of a spaced seed are less related than the hits of a contiguous word; for a same specificity (number of hits), the sensitivity is better.

■ **EXAMPLE 15.3 Spaced Seeds**

With the spaced seed ##-## of weight 4, the nucleic key AA-CT matches the four strings AAACT, AACCT, AAGCT and AATCT. This seed is lossless on $(40, 2, 10)-$Hrepeats: that means that all $(40, 2, 10)-$Hrepeats share at least a gapped 4-gram shaped by ##-##. This is better than the contiguous 4-gram #### that misses some (rare) alignments.

On $(40, 2, 20)-$Hrepeats, both seeds are lossy. On a Bernoulli model, the spaced seed ##-## has now a sensibility of 86,8%, whereas the seed #### only achieves a 79,8% sensibility. Sensibilities are computed with Iedera [27].

The idea of using spaced seeds for biological sequence comparisons first was proposed in 2002 by Ma *et al.* [37] in the PatternHunter software. Following this article, theoretical design and usage of better seeds became an active field of research [6, 11, 32, 38], with extensions on vector seeds [3], protein seeds [4, 24], and subset seeds [27, 53]. The most complete and recent survey of this domain is [5].

In all those seed models, one *designs* appropriate seeds according to sensitivity/selectivity criteria and the class of target alignments. Moreover, instead of using a single seed, one can use several seeds simultaneously (so-called *multiple seeds*) to improve further the sensitivity/selectivity trade-off.

### 15.4.3 Latencies and Neighborhood Indexing

***15.4.3.1 Latencies for the Finishing Step.*** What exactly is stored during the indexing phase? For each key, we want to remember the list of all its occurrences in the database. In the usual *offset indexing* approach, depicted on Figure 15.3, an offset of $\log N$ bits is stored for each seed position (where $N$ is the size of the database). The index size thus is equal to $N \times \log N$ bits.

For each query position, each hit returned by the filtering phase leads to an iteration of the finishing phase. This iteration accesses some neighborhood of the positions. These memory accesses are *random*, that is, unpredictable and noncontiguous.

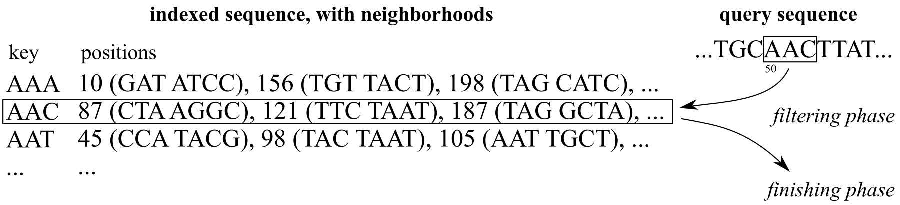

indexed sequence, with neighborhoods
key
positions
AAA 10 (GAT ATCC), 156 (TGT TACT), 198 (TAG CATC), ...
AAC 87 (CTA AGGC), 121 (TTC TAAT), 187 (TAG GCTA), ...
AAT 45 (CCA TACG), 98 (TAC TAAT), 105 (AAT TGCT), ...
... ...
query sequence
...TGCAACTTAT...
50
filtering phase
finishing phase

**Figure 15.4** In the neighborhood indexing approach, for each key, a small neighborhood of each key occurrence is stored redundantly in the index. Here $L = 7$ nucleotides are stored with each position. The filtering phase needs one memory access (per key). The finishing phase does not need any more additional access.

Such accesses are not cached efficiently and require high latencies [19]. This is especially true when using multiple seeds, for example, in the BlastP approach (on average, 26 index look-ups for the Blosum-62 background distribution of amino acids [48]).

***15.4.3.2 Neighborhood Indexing.*** A way to reduce the computation time thus is to avoid as far as possible such random memory accesses. In [48], a *neighborhood indexing* approach was proposed. The idea is to store additionally, for each key occurrence, its left and right neighborhoods in the sequence (Figure 15.4). Thus, given a position in the query and its corresponding key, all neighborhoods of this key occurrences in the database are obtained through a single memory access. There is no need for further memory accesses to random positions in the original sequence. The overall index size then is equal to $N \times (\log N + \alpha L)$ bits, where $\alpha$ is the number of bits for coding a character (nucleotide or amino acid), and $L$ is the total length of the neighborhoods.

The main advantage of the neighborhood indexing is that it speeds the execution time by a factor ranging between 1.5 and 2 over the offset indexing [48]. The actual speed gain depends on the database length and on many implementation and architecture parameters (such as memory and cache sizes, cache strategies, and access times) that will not be discussed here. An obvious drawback of the neighborhood indexing is the additional memory it requires to store neighborhoods. Comparing the two indexing schemes, the ratio between the overall index sizes of the neighborhood indexing and the offset indexing is $1 + \alpha L / \log N$. Usual values for $\log N$ are between 20 and 40, and usual values for $L$ are between between $2 \times 20$ and $2 \times 200$; hence, this ratio is between 2 and 21.

***15.4.3.3 Implementation Detail of the Neighborhood Indexing: Alphabet Reduction.*** In [48], a reduction of this ratio for the proteic case is proposed. The idea is is to use a reduced amino acid alphabet, and thus to reduce $\alpha$. Grouping amino acids was studied in several papers [8, 13, 33, 42]. Groups can rely on amino acid physical-chemical properties or on a statistical analysis of alignments. For example, the authors of [42] computed correlation coefficients between pairs of amino acids based on the BLOSUM50 matrix and used a greedy algorithm to merge them.

A branch-and-bound algorithm for partitioning the amino acids was proposed in [8]. An extreme application of the grouping gives alphabets with only two symbols. Li *et al.* proposed in [33] the alphabet $\Sigma_2 = \{\text{CFYWMLIV, GPATSNHQEDRK}\}$. Using this alphabet for amino acids divides the storage requirement per 5 but is, of course, far less precise.

In [48], we showed that we could retrieve the original sensitivity by using longer neighborhoods and keeping the original alphabet for the query string. We computed *rectangular* BLOSUM matrices (ReBlosum, Example 15.4) that compare amino acid groups (from the indexing alphabet) with actual amino acids (for the query string). Our method applies on any amino acid partition. The best results were obtained with Li's $\Sigma_2$ alphabet; with the same sensibility, 35% less memory is neeeded for a neighborhood length of $2 \times 32$ instead of $2 \times 11$ amino acids when $\log N = 24$.

**■ EXAMPLE 15.4 Alphabet Reduction**

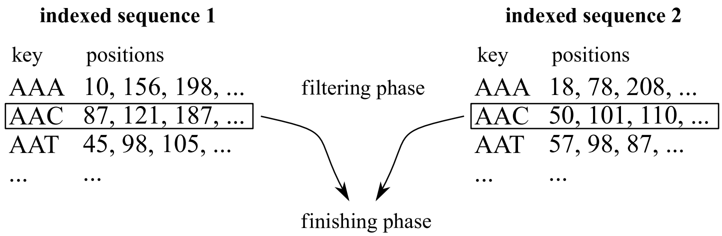
| | C | F | Y | W | M | L | I | V | G | P |
|---|---|---|---|---|---|---|---|---|---|---|
| CFYWMLIV | 4 | 4 | 3 | 4 | 3 | 4 | 4 | 3 | −6 | −6 |
| GPATSNHQEDRK | −4 | −5 | −4 | −6 | −3 | −5 | −5 | −4 | 2 | 2 |
| | A | T | S | N | H | Q | E | D | R | K |
| CFYWMLIV | −2 | −2 | −4 | −6 | −4 | −4 | −6 | −7 | −5 | −5 |
| GPATSNHQEDRK | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 |

The ReBlosum matrix for comparison of Li's alphabet $\Sigma_2$ (indexing alphabet) with the usual alphabet $\Sigma_{20}$ (querying alphabet) computed on alignments with 62% identity. On a Bernoulli model, comparing a query of length 32 (on $\Sigma_{20}$) with an indexed neighborhood of length 32 (on $\Sigma_2$) is as sensible as comparing a query of length 11 (on $\Sigma_{20}$) with a usual neighborhood on length 11 on $\Sigma_{20}$ but with a 35% memory reduction.

### 15.4.4 Seed-Based Heuristics Implementations

***15.4.4.1 CPU Implementations.*** The first widely used implementations of seed-based heuristics were Fasta [46] then Blast [1]. The Blast 2 implementation [2] uses a double hit, allowing a greater sensibility.

PatternHunter [37], followed by PatternHunter 2 [31], are the reference papers for the spaced seeds and the multiple seeds ideas. Another widely used tool, Blat [23], allows one mismatch in a contiguous seed and, thus, is equivalent to a collection of several spaced seeds. Yass [45, 44] is a comparison tool between nucleic sequences that implements several optimisation to seeds, including multiple "transition constrained" seeds, favoring transition mutations between purines and between pyrimidines. Yass come with the Iedera tool [27] that designs and evaluates various types of seeds.



**indexed sequence 1**

| key | positions |
|---|---|
| AAA | 10, 156, 198, ... |
| AAC | 87, 121, 187, ... |
| AAT | 45, 98, 105, ... |
| ... | ... |

**indexed sequence 2**

| key | positions |
|---|---|
| AAA | 18, 78, 208, ... |
| AAC | 50, 101, 110, ... |
| AAT | 57, 98, 87, ... |
| ... | ... |

**Figure 15.5** ORIS and PLAST seed-based indexing. As the two sequences are indexed, the filtering phase is a simultaneous scan of both indexes. If the neighborhoods also are included in the index, then here there is almost no more random access during both filtering and finishing phases.

A really nice idea appeared in 2008 in ORIS and then PLAST [28, 43]. The idea here is to build two indexes, one for each bank and then to scan simultaneously the both indexes (Figure 15.5). The advantage of such an approach is that there is virtually no cache miss for seeds that are not extended further; all latencies are removed even during the filtering step.

Seed-based heuristics are found in wider applications, for example, in ZOOM, a read mapper for high-throughput sequencers [18].

***15.4.4.2 Parallel Implementations.*** Instead of a serial implementation of a seed-based heuristics, one can parallelize some parts of the computation. Fine-grained parallelism can be realized either through vector single instruction multiple data (SIMD) instructions or on specialized architectures (custom processors or reconfigurable FPGA processors). Moreover, threads on a multicore architecture, or clusters of any of the previous solutions can provide additional coarse-grained parallelism.

- *Blast-like seeds.* The first hardware implementation of a seed-based heuristic was done in 1993 on a custom processor with the BioSCAN architecture [57]. Several implementations on reconfigurable field-programmable gate array (FPGA) processors have been developed independently since 2003 ([9, 20, 26, 41], see [29] for a review). A cluster-enabled version of Blast was proposed on message possing interface (MPI) [12, 59].
- *Other heuristics.* DASH [25] is an algorithm implemented on FPGA with a better sensibility than BLAST. In 2006, [30] proposed an architecture designed to proteic comparisons using seeds of three amino acids. We proposed the implementation on subset seeds [27] on a FPGA architecture coupled with large Flash memories [47]. The PLAST algorithm was designed especially for easy parallelization, and several parallel versions are available [43].

Finally, the new many-cores architectures like graphic processing units (GPUs) also can offer both levels of parallelism. Recently, numerous parallel GPU implementations of regular Smith–Waterman dynamic programming were proposed

[34, 35, 36, 39, 55]. Seed-based heuristics also could take benefit from those architectures.

## 15.5 CONCLUSION

Sequence homologies are the key to molecular biology studies. Modeled by approximate repeats and approximate multiple repeats, the discovery of homologies remains a difficult task suffering from time bottlenecks. Moreover, molecular biology studies have to cope with datasets whose size grows exponentially. Today biologists often must restrict the area of their research while looking for similarities into sequences or between sequences. Some computations, in particular concerning the research of similarities not limited to two occurrences, are simply unfeasible.

Given a similarity model, filters are methods permitting quick focus on sequences fragments that may contain some repeats occurrences. Thus, after a filtering step, programs designed for the similarity research may be applied to much smaller datasets and find faster repeat occurrences. This chapter exposed two kind of filters. First, lossless filters that ensure that no occurrences of repeats may be filtered out. Second, lossy seed filters that are heuristics that do not assure that no repeat occurrences are missed after filtration.

Used as first steps of similarity research, *lossless filters* present the large advantage that they do not modify the results quality. Although applied for the research of multiple repeats, they can reduce computation time by several orders of magnitude. However this kind of filter generally is limited to the research of well-conserved similarities. On the other hand, *lossy seed filters* are methods widely used that provide even faster results. This kind of filter may be used for finding similarities with a high divergence rate. However, even if some approaches estimate the sensitivity of the results, they cannot ensure finding 100% of the searched results.

Today, new sequencing techniques increase even the exponential rate of the flood of data. New methods must be designed to improve further the similarity search still working for increasing the sensitivity close to the full sensitivity.

## 15.6 ACKNOWLEDGMENTS

We are grateful to Julien Allali (Université Bordeaux 1) and Jérémie Bourdon (Université de Nantes) for their comments on this chapter.

## REFERENCES

1. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, 1990.
2. S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, 1997.

3. B. Brejová, D. Brown, and T. Vinar. Vector seeds: An extension to spaced seeds. *J Comput Syst Sci*, 70(3):364–380, 2005.
4. D. Brown. Optimizing multiple seeds for protein homology search. *IEEE Trans Comput Biol Bioinformatics*, 2(1):29–38, 2005.
5. D.G. Brown. A survey of seeding for sequence alignment. In I. Mandoiu and A. Zelikovsky, editor, *Bioinformatics Algorithms: Techniques and Applications*. Wiley-Interscience, New York, 2008, pp. 126–152.
6. J. Buhler, U Keich, and Y. Sun. Designing seeds for similarity search in genomic DNA. *J Comput Syst Sci*, 70(3):342–363, 2005.
7. S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. *q*-gram based database searching using a suffix array (QUASAR). *Annual Conference on Research in Computational Molecular Biology (RECOMB 99)*, Lyon, France, 1999, pp. 77–83.
8. N. Cannata, S. Toppo, C. Romualdi, and G. Valle. Simplifying amino acid alphabets by means of a branch and algorithm and substitution matrices. *Bioinformatics*, 18(8):1102–1108, 2002.
9. C. Chang. BLAST implementation on BEE2. University of California at Berkeley, 2004.
10. M. Crochemore, G.M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM J Comput*, 32(6):1654–1673, 2003.
11. M. Csürös and B. Ma. Rapid homology search with two-stage extension and daughter seeds. *International Computing and Combinatorics Conference (COCOON 05)*, Kunming, China, 2005, pp. 104–114.
12. A. Darling, L. Carey, and W. Feng. The design, implementation, and evaluation of mpi-BLAST. *ClusterWorld Conference and Expo (CWCE 2003)*, SanJose, CA, 2003.
13. R.C. Edgar. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Res*, 32(1):380–385, 2004.
14. A. Fontaine, A. de Monte, and H. Touzet. MAGNOLIA: multiple alignment of protein-coding and structural RNA sequences. *Nucleic Acids Res*, 36(S2):W14–W18, 2008.
15. M.C. Frith, N.F.W. Saunders, B. Kobe, and T.L. Bailey. Discovering sequence motifs with arbitrary insertions and deletions. *PLoS Comput Biol*, 4(5):e1000071, 2008.
16. M.C. Frith, U. Hansen, J.L. Spouge, and Z. Weng. Finding functionnal sequence elements by multiple local alignement. *Nucleic Acid Res*, 32(1):189–200, 2004.
17. R. Grossi and F. Luccio. Simple and efficient string matching with k mismatches. *Inf Proces Lett*, 33(3):113–120, 1989.
18. L. Hao, Z. Zefeng, Q.Z. Michael, M. Bin, and L. Ming. ZOOM! Zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437, 2008.
19. J.L. Hennessy and D.A. Patterson. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, Burlington, MA, 2006.
20. A. Jacob, J. Lancaster, J. Buhler, and R. Chamberlain. FPGA-accelerated seed generation in Mercury BLASTP. *Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 07)*, Napa Valley, CA, 2007, pp. 95–106.
21. R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J Res Dev*, 31(2):249–260, 1987.
22. R.M. Karp, S. Shenker, and C.H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans Database Syst*, 28:51–55, 2003.

23. W. James Kent. BLAST - the BLAST-like alignment tool. *Genome Res*, 12:656–664, 2002.

24. D. Kisman, M. Li, B. Ma, and W. Li. tPatternHunter: gapped, fast and sensitive translated homology search. *Bioinformatics*, 21(4):542–544, 2005.

25. G. Knowles and P. Gardner-Stephen. A new hardware architecture for genomic and proteomic sequence alignment. *IEEE Computational Systems Bioinformatics Conference (CSBC 04)*, Stanford, CA, 2004.

26. P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, and J. Lancaster. Biosequence similarity search on the Mercury system. *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 04)*, Galvaston, TX, 2004.

27. G. Kucherov, L. Noé, and M. Roytberg. A unifying framework for seed sensitivity and its application to subset seeds. *J Bioinformatics Comput Biol*, 4(2):553–569, 2006.

28. D. Lavenier. Ordered index seed algorithm for intensive dna sequence comparison. *IEEE International Workshop on High Performance Computational Biology (HiCOMB 08)*, Miami, FL, 2008.

29. D. Lavenier and M. Giraud. Reconfigurable Computing. *Bioinformatics Applications*. Springer, New York, 2005.

30. D. Lavenier, L. Xinchun, and G. Georges. Seed-based genomic sequence comparison using a FPGA/FLASH accelerator. *Field Programmable Technology (FPT 2006)*, Bangkok, Thailand, 2006, pp. 41–48.

31. M. Li, B. Ma, D. Kisman, and J. Tromp. PaternHunter II: Highly sensitive and fast homology search. *Genome Inform*, 14:164–175, 2003.

32. M. Li, M. Ma, and L Zhang. Superiority and complexity of the spaced seeds. *Symposium on Discrete Algorithms (SODA 06)*, Miami, FL, 2006, pp. 444–453.

33. T.P. Li, K. Fan, J. Wang, and W. Wang. Reduction of protein sequence complexity by residue grouping. *Protein Eng*, 16(5):323–330, 2003.

34. L. Ligowski and W. Rudnicki. An efficient implementation of Smith-Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. *IEEE International Workshop on High Performance Computational Biology (HiCOMB 09)*, Rome, Italy, 2009.

35. W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig. GPU-ClustalW: Using graphics hardware to accelerate multiple sequence alignment. *IEEE International Conference on High Performance Computing (HiPC 06)*, volume 4297 of *Lecture Notes in Computer Science (LNCS)*, 2006, pp. 363–374.

36. Y. Liu, D. Maskell, and B. Schmidt. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Res Notes*, 2(1):73, 2009.

37. B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

38. D. Mak, Y. Gelfand, and G. Benson. Indel seeds for homology search. *Bioinformatics*, 22(14):e341–e349, 2006.

39. S.A Manavski and G. Valle. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, 9 Suppl 2:S10, 2008.

40. E.R. Mardis. Next-generation DNA sequencing methods. *Annu Rev Genom Hum Genet*, 9(1):387–402, 2008.

41. K. Muriki, K. Underwood, and R. Sass. RC-BLAST: Towards an open source hardware implementation. *IEEE International Workshop on High Performance Computational Biology (HiCOMB 05)*, Denver, CO, 2005.

42. L.R. Murphy, A. Wallqvist, and L. Ronald. Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Eng*, 13(3):149–152, 2000.

43. V.H. Nguyen and D. Lavenier. PLAST: Parallel local alignment search tool. *BMC Bioinformatics*, To appear.

44. L. Noé and G. Kucherov. Improved hit criteria for DNA local alignement. *Bioinformatics*, 5(149), 2004.

45. L. Noé and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Res*, 33:W540–W543, 2005.

46. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci.*, 85:3244–3248, 1988.

47. P. Peterlongo, L. Noé, D. Lavenier, G. Georges, J. Jacques, G. Kucherov, and M. Giraud. Protein similarity search with subset seeds on a dedicated reconfigurable hardware. *Parallel Biocomputing Conference (PBC 07)*, volume 4967 of *Lecture Notes in Computer Science (LNCS)*, 2007.

48. P. Peterlongo, L. Noé, D. Lavenier, V.H. Nguyen, G. Kucherov, and M. Giraud. Optimal neighborhood indexing for protein similarity search. *BMC Bioinformatics*, 9(534), 2008.

49. P. Peterlongo, N. Pisanti, F. Boyer, A. Pereira do Lago, and M.-F. Sagot. Lossless filter for multiple repetitions with hamming distance. *J Discrete Algorithm*, 6(3):497–509, 2008.

50. P. Peterlongo, N. Pisanti, F. Boyer, and M.-F. Sagot. Lossless filter for finding long multiple approximate repetitions using a new data structure, the bi-factor array. *International Symposium on String Processing Information Retrieval (SPIRE 05)*, Buenos Aries, Argentina, 2005, pp. 179–190.

51. P. Peterlongo, G.A.T. Sacomoto, A. Pereira do Lago, N. Pisanti, and M.-F. Sagot. Lossless filter for multiple repeats with bounded edit distance. *BMC Algorithm Mol Biol*, 4(3), 2009. To appear.

52. K.R. Rasmussen, J. Stoye, and E.W. Myers. Efficient $q$-gram filters for finding all $\epsilon$-matches over a given length. *J Comput Biol*, 13(2):296–308, 2006.

53. M. Roytberg, A. Gambin, L. Noé, S. Lasota, E. Furletova, E. Szczurek, and G. Kucherov. Efficient seeding techniques for protein similarity search. *Proceedings of Bioinformatics Research and Development (BIRD 08)*, volume 13 of *Communications in Computer and Information Science*, 2008, pp. 466–478.

54. F. Sanger, S. Nicklen, and A.R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proc Ntl Acad Sci U S A*, 74(12):5463–5467, 1977.

55. M.C Schatz, C. Trapnell, A.L. Delcher, and A. Varshney. High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, 8:474, 2007.

56. J. Shendure and H. Ji. Next-generation DNA sequencing. *Nat Biotechnol*, 26(10):1135–1145, 2008.

57. R.K. Singh, S.G. Tell, C.T. White, D. Hoffman, V.L. Chi, and B.W. Erickson. A scalable systolic multiprocessor system for analysis of biological sequences. *Symposium on Research on Integrated Systems*, Seattle, WA, 1993, pp. 168–182.

58. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147:195–197, 1981.

59. O. Thorsen, B. Smith, C.P. Sosa, K. Jiang, H. Lin, A. Peters, and W. Fen. Parallel genomic sequence-search on a massively parallel system. *International Conference on Computing Frontiers (CF 07)*, Las Vegas, NV, 2007.

60. E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor Comput Sci*, 92(1):191–211, 1992.

# 16

# NOVEL COMBINATORIAL AND INFORMATION-THEORETIC ALIGNMENT-FREE DISTANCES FOR BIOLOGICAL DATA MINING

Chiara Epifanio, Alessandra Gabriele, Raffaele Giancarlo,
and Marinella Sciortino

## 16.1 INTRODUCTION

Sequence comparison plays a central role in many scientific areas [70] and, in particular, for computational molecular biology [41, 81]. In fact, biological molecules can be seen as linear sequences of discrete units similar to linguistic representations, despite their physical nature as a three-dimensional (3D) structure and the dynamic nature of molecular evolution.

The problem of defining good mathematical functions to measure similarity between biological sequences is fundamental for their analysis. Indeed, a high similarity among biological sequences, as measured by mathematical functions, usually gives a strong indication of functional relatedness and/or common ancestry [41]. However, there is not a bijective correspondence between sequence and structure or sequence and function because, in general, the converse of the previous statement is not true.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Classically, the notions of similarity and distance hinge on sequence alignment [41]. Algorithmically, these methods usually are implemented by using dynamic programming and target specific goals such as global and local alignments, with many of their variants (*e.g.*, [40,41,60,72]). Because of their worst case superlinear running time in the input parameters, these methods are considered inadequate for the analysis of long sequences in which one usually resorts to heuristic algorithms, such as BLAST [13,14] and FASTA [64,65]. In case of protein alignment, all algorithms we have mentioned so far use matrices point accepted mutation (PAM) [27] and blocks of amino acid substitution matrix (BLOSUM) [42] as their scoring schemes [41].

Now that entire genomes are available, the sequence alignment approach no longer is perceived as right for sequence comparison. In fact, it is not suitable to measure events and mutations that involve very long segments of genomic sequences because sequence alignment considers only local mutations of the genome. Actually, the conservation of contiguity underlying alignment is at odds with genetic recombination, which includes shuffling subgenomic DNA fragments. This is the case, for instance, of orthologous regulatory sequences or no orthologous functionally related sequences (such as cis-regulatory modules), which do not show any statistically significant alignment. Moreover, with reference to proteomic sequences comparison, the use of scoring matrices in the alignment-based methods shows limitations when one has to deal with sequences with less than 20% sequence identity. Furthermore, the growing mass of biological data makes impracticable the alignment-based approach both for running time and memory usage.

The need to overcome these critical limitations of alignment-based measures, in whole or in part, has required the development of alternative approaches. The first systematic organization of alignment-free measures, together with the name, was given in the ground-breaking paper by Vinga and Almeida in [78].

Most of the alignment-free methods can be classified in accordance with well-established paradigms. A first paradigm concerns information-theoretic approaches that use classical information measures. The power of information measures is a result of their ability to determine statistical dependence among sequences, which in turn, may reveal biologically important relationships. Another paradigm regards the use of techniques and notions from combinatorics on words. The third paradigm collects methods that are based on the notion of subword composition of sequences in which vectors of word frequencies of a given length are considered, whereas the fourth paradigm focuses on counting the exact matches of words of a given length.

Within those paradigms, we have identified some representative methods that have been introduced recently and that provide promising solutions for alignment-free comparison and that will be the object of this chapter. Moreover, we also present several domains of biological relevance in which experiments support the conclusion that the methods performed here are indeed adequate. As for algorithm experimentation and engineering, we provide a kernel of datasets and publicly available software libraries that can be used for benchmarking and comparative studies.

The remainder of the chapter is organized as follows: Section 16.2 presents some recent and innovative alignment-free methods that make explicit use of information measures, based on the notions of empirical relative entropy and empirical mutual information between sequences. Section 16.3 is devoted to the presentation of some combinatorial alignment-free methods chosen from the ones available in this class for either having been used already for large-scale, genome-wide, studies or having the potential to perform well in those studies. In Section 16.4 we describe some alignment-free methods based on the notion of subword composition of sequences, which, thanks to extensive experimentation, have become significant for biological data mining. Section 16.5 presents some alignment-free word matches methods and, in particular, the distance $D_2$, which stands out for its mathematical elegance as well as its usefulness and two generalizations of it. The first one is a straightforward extension of $D_2$ to approximate $k$-word matches, whereas the second one, nicknamed $D_2z$, is a variant of $D_2$, which offers several advantages with respect to it.

Section 16.6 contains the main experimental contributions to the bioinformatic area of the papers cited in Sections 16.2 to 16.5. In particular, we briefly present the biological domains in which the distances described here have been applied. Those domains are representative of the use of alignment-free distances in biological investigations.

Moreover, Section 16.7 presents a kernel of datasets that seem to be the most used for benchmarking purposes. Such datasets are of interest for comparative analysis of similarity measures, as they are used for classification and phylogenetic studies. Furthermore, we provide the most prominent publicly available software libraries that one can use for benchmarking and comparative studies. The final section is devoted to conclusions.

## 16.2 INFORMATION-THEORETIC ALIGNMENT-FREE METHODS

Shannon information theory often is perceived as being of interest for data communication and storage. However, it is also deeply related to classification, data mining, and analysis. For many years, in computational biology and bioinformatics, the most successful use of information theory has been for sequence analysis and, in particular, to measure the “deviation from randomness” of nucleotide and amino acid sequences. Anyway, the power of information measures is because of their ability to determine statistical dependence among sequences, which in turn, may reveal biologically important relationships. A homogeneous presentation of the role of information theory and data compression in computational biology is given in [38]. It is also natural that those techniques explicitly offer natural alignment-free methods for biological data analysis. Moreover, fundamental measures of information are also at the very foundation of many other alignment-free methods. That is, the former are not used explicitly to compute the latter, but they are connected conceptually to them. For this reason, the first subsection is devoted to outline the relation between information measures and “similarity” of sequences, via statistical dependency. The remaining subsections are devoted to some recent and particularly innovative alignment-free methods that make explicit use of information measures. In particular, the second subsection presents two measures based on empirical relative entropy between sequences; the first one uses, in a novel way, the empirical version of entropy, whereas the second introduces the use of Markov chains, learned from the two sequences for which similarity is to be assessed to evaluate entropy. The third subsection is devoted to an alignment-free method that is based on the notion of empirical mutual information between sequences. The interesting novelty of that approach consists of the introduction of a test to quantify the level of statistical significance of a dependency between two sequences as quantified by an information measure.

### 16.2.1 Fundamental Information Measures, Statistical Dependency, and Similarity of Sequences

Three basic concepts underlying Shannon information theory are certainly entropy, relative entropy, and mutual information [25]. They are presented here in their generality, whereas Subsections 16.2.2 and 16.2.3 consider their empirical counterparts as they are applied to the design of alignment-free methods.

The entropy of a random variable is a measure of its uncertainty. Technically, it quantifies the amount of information (in bits) required on average to describe the random variable. More precisely, given a discrete random variable $X$ with alphabet $\Sigma$ and probability mass function $p(x) = \Pr\{X = x\}$, $x \in \Sigma$, the *entropy* $H(X)$ is defined by

$$H(X) = -\sum_{x \in \Sigma} p(x) \log p(x)$$

Note that entropy is a functional of the probability distribution of $X$ (*i.e.*, it is dependent only on the probabilities and not on the actual values taken by that random variable).

The *relative entropy* or *Kullback–Leibler distance* is a measure of the distance between two probability distributions. It is defined between two probability mass functions $p(x)$ and $q(x)$ as

$$\mathrm{RE}(p, q) = \sum_{x \in \Sigma} p(x) \log \frac{p(x)}{q(x)}$$

The relative entropy is a measure of the inefficiency of assuming that the distribution is $q$ when the true distribution is $p$. It is always nonnegative and it is zero if and only if $p = q$. It also can be interpreted as a quantification of how $p$ is "distant" from $q$. Technically, however, it is not a distance measure because it is not symmetric and does not satisfy the triangle inequality.

Mutual information of two random variables $X$ and $Y$ is a measure of the amount of information that the first variable has about the second one. It also can be seen as a special case of the relative entropy of two probability distributions. Indeed, given two random variables $X$ and $Y$ with a joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$, the *mutual information* is the relative entropy between the joint distribution and the product distribution $p(x)p(y)$, that is,

$$I(X,Y) = \mathrm{RE}(p(x,y), p(x)p(y)) = \sum_{x,y\in\Sigma} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

The following result is fundamental for the entire chapter:

**Theorem 16.1** *Given two random variables $X$ and $Y$, $I(X,Y) \geq 0$, with equality if and only if $X$ and $Y$ are statistically independent.*

The statement establishes a deep connection between statistical dependency of random variables and an information measure; we can quantify the former by computing the latter, with the following great practical impact. Statistical dependency on real data can be determined only once that the probability distributions are estimated, usually a rather complex and error-prone procedure. When one resorts to the assessment of statistical dependency via information measures, an entire world of heuristic solutions becomes available, via the connection of information measures to data compression and combinatorics on words. Finally, the estimation of statistical dependency is fundamental for computational biology because the statistical dependency among sequences commonly is accepted as an indication of biological relatedness.

### 16.2.2 Methods Based on Relative Entropy and Empirical Probability Distributions

Throughout this chapter, let $A = a_1a_2\cdots a_n$ and $B = b_1b_2\cdots b_m$ denote two strings over the alphabet $\Sigma$. Fix an integer $k \geq 1$, which we refer to as the *word size*. Moreover, we refer to any string in $\Sigma^k$ as a *$k$-word.*

Consider string $A$ and, for each $k$-word $w_i$ in $\Sigma^k$, let $n_{k,i}$ be the number of occurrences of $w_i$ in $A$. Let $\{p_{k,i} = n_{k,i}/(|A| - k + 1)\}_{i=1}^{|\Sigma|^k}$ be the $k$th *order empirical probability distribution* for the string $A$. The $k$th *order empirical entropy* of the string $A$ then is defined as

$$H_k(A) = -\sum_{i=1}^{|\Sigma|^k} p_{k,i}\log(p_{k,i})$$

It is important to stress that there is an important difference between the entropy defined in the probabilistic setting and its empirical counterpart. In fact, Shannon entropy is an expected value taken on a set of strings, whereas empirical entropy is defined "pointwise" for each string, and it can be used as a function of the string structure without any assumption on the information source emitting the string.

Similarly, it is possible to extend the concept of relative entropy to empirical probability distributions based on $k$-words. As a result, $k$-word empirical relative entropy becomes a natural approach to measure dissimilarity between biological sequences. As pointed out by Vinga and Almeida [78], the case $k = 1$ (*i.e.*, the empirical Kullback-Leibler distance), has been one of the first methods used in this area. Apparently however, the case $k > 1$ has been studied experimentally only recently by Wu *et al.* [85]. Technically, consider $A$ and $B$ and let $\{p^A_{k,i}\}$ and $\{p^B_{k,i}\}$ be their $k$-word empirical probability distributions, respectively. The $k$th *order empirical relative entropy* between $A$ and $B$ is defined as

$$\mathrm{RE}_k(A, B) = \sum_{i=1}^{|\Sigma|^k} p^A_{k,i} \log_2 \left( \frac{p^A_{k,i}}{p^B_{k,i}} \right)$$

To avoid an indefinite value when $p^B_{k,i} = 0$, Wu *et al.* also suggest to modify that formula by adding a unit to both terms of the probability ratio.

It is obvious that the representation of biological sequences by their empirical probability distributions and information measures involves some trade-offs. When $k$ is small, one loses information about the sequences (*i.e.*, even very dissimilar sequences can have probability distributions close to each other). On the other hand, as $k$ increases, $k$-word information measures suffer of the so-called *finite sample effect* yielding degenerate results [38]; when $k$ is large, we do not have enough substrings in the strings to sample accurately the "true" probability distributions underlying the strings. In particular, some word types that are present in one string may be absent in the other. To date, no satisfactory solution to this problem is known, although one common approach consists of grouping some $k$-words having frequencies up to a particular threshold into one event, but this is accompanied by a loss of sequence information.

Wang and Zheng [80] propose a novel method, referred to as *Weighted Sequence Entropy* (WSE) that attempts to overcome the finite sample effect problem. Although it is based on the classical empirical relative entropy, the two distributions involved are $\{p^A_{k,i}\}$ and the arithmetical average value of distributions $\{p^A_{k,i}\}$ and $\{p^B_{k,i}\}$. Wang and Zheng also introduce two slight variations of that measure by assigning weights to $\{p^A_{k,i}\}$ and $\{p^B_{k,i}\}$. These variants are equivalent to the $k$-word relative entropy when $k$ is small but tend to avoid the degeneration when $k$ increases. Some technical details follow.

Consider the probability distributions $\{p^A_{k,i}\}$, $\{(p^A_{k,i} + p^B_{k,i})/2\}$, and their relative entropy

$$\mathrm{RE1}_k(A, B) = \sum_{i=1}^{|\Sigma|^k} p^A_{k,i} \log_2 \left( \frac{2p^A_{k,i}}{p^A_{k,i} + p^B_{k,i}} \right)$$

Because, by definition, $\mathrm{RE1}_k(A, B) \neq \mathrm{RE1}_k(B, A)$, to ensure the symmetry condition, Wang and Zheng consider the sum $d_1(A, B) = \mathrm{RE1}_k(A, B) + \mathrm{RE1}_k(B, A)$.

By using the empirical information entropy, $d_1(A, B)$ can be rewritten as

$$d_1(A, B) = 2H\left(\frac{A+B}{2}\right) - H(A) - H(B)$$

Notice that $(A+B)/2$ is a shorthand for the probability distribution $\{(p^A_{k,i} + p^B_{k,i})/2\}$. It is interesting to point out that the "averaging" method just outlined generalizes to give weights to the probability distributions $\{p^A_{k,i}\}$ and $\{p^B_{k,i}\}$ as the following two examples illustrate. Let

$$\mathrm{RE2}_k(A, B) = \sum_{i=1}^{R} p^A_{k,i} \log_2\left(\frac{(n+m)p^A_{k,i}}{np^A_{k,i} + mp^B_{k,i}}\right)$$
$$\mathrm{RE3}_k(A, B) = \sum_{i=1}^{R} p^A_{k,i} \log_2\left(\frac{(\sqrt{n}+\sqrt{m})p^A_{k,i}}{\sqrt{n}p^A_{k,i} + \sqrt{m}p^B_{k,i}}\right)$$

Their corresponding distance measures are as follows:

$$d_2(A, B) = (n+m)H\left(\frac{nA+mB}{n+m}\right) - nH(A) - mH(B)$$
$$d_3(A, B) = (\sqrt{n}+\sqrt{m})H\left(\frac{\sqrt{n}A+\sqrt{m}B}{\sqrt{n}+\sqrt{m}}\right) - \sqrt{n}H(A) - \sqrt{m}H(B)$$

It is of interest to point out that, because $d_1$, $d_2$, and $d_3$ are linear combinations of empirical entropies, this approach has been nicknamed as the weighted sequence entropy. Notice that none of the distances can have an indefinite value because they are all nonnegative and bounded by 2, $n+m$, and $\sqrt{n}+\sqrt{m}$, respectively.

In the case of short words sizes, Wang and Zheng show that their three revised versions of the classical empirical relative entropy perform as well as empirical relative entropy in phylogenetic inferences. However, for large word sizes, their methods still grant reliable phylogenetic inferences, whereas relative entropy returns degenerate results because of the absence of some word types. Moreover, the authors compare, with positive results, their new methods with other existing ones for the construction of phylogenetic trees, namely, Euclidean distance [18, 78], linear correlation coefficient [32,66], cosine function [68,69,73,74], and information-based similarity index [43].

Dai *et al.* [26] propose two other methods, whose novelty is to combine the statistical information about $k$-words by deriving it both empirically and from a Markov chain that supposedly represents well the salient features of a family of strings. The rationale is that both $k$-word distributions and Markov models contain important information about strings, but they are of a different nature. In $k$-word distributions, the probability of each word is dictated punctually by the string, whereas it is an average, or the result of a training process on a set of strings, when one uses a Markov chain.

The first two measures Dai *et al.* introduce, denoted $rre.k.r$ and $S_1.k.r$, extend the concept of empirical relative entropy between strings to Markov chains of order $r$. The other ones, denoted $wre.k.r$ and $S_2.k.r$, are defined starting from $rre.k.r$. In particular, for these last two measures the probability of a $k$-word is computed by multiplying the probability of that $k$-word, as predicted by a Markov chain of order $r$, by the same probability obtained via the frequency of occurrence of that $k$-word in the string. We provide some details next.

The *revised relative entropy* $rre.k.r$ is an information measure between two Markov chains $M_A^r$ and $M_B^r$ of order $r$. Each Markov chain is learned from $A$ and $B$, respectively. It is

$$rre.k.r(M_A^r, M_B^r) = \sum_{w \in \Sigma^k} \left| p(w|M_A^r) \ln \left( \frac{2p(w|M_A^r)}{p(w|M_A^r) + p(w|M_B^r)} \right) \right|$$

where $p(w|M^r)$ denotes the probability of the $k$-word $w = w_1 w_2 \cdots w_k$ as computed via a Markov chain $M^r$ of order $r$. In particular, for $0 \leq r \leq 2$, $p(w|M^r)$ can be computed as follows

$$p(w|M^r) = \begin{cases} \pi_{w_1}\pi_{w_2} \cdots \pi_{w_k} & \text{if } r = 0 \\ \pi_{w_1} p(w_1, w_2) p(w_2, w_3) \cdots p(w_{k-1}, w_k) & \text{if } r = 1 \\ \pi_{w_1 w_2} p(w_1 w_2, w_3) p(w_2 w_3, w_4) \cdots p(w_{k-2} w_{k-1}, w_k) & \text{if } r = 2 \end{cases}$$

where $p(i, j)$ is obtained from the state transition matrix of $M^r$. The vector $\pi_i = p(w_1 = S_i)$ is the initial state probability distribution, with $S_i$ denoting the $i$th state of $M^r$.

The symmetric form of $rre.k.r$, denoted by $S_1.k.r$, is defined by

$$S_1.k.r(M_A^r, M_B^r) = \begin{cases} 0 & \text{if } M_A^r = M_B^r \\ \sum_{w \in \Sigma^k} p(w|M_A^r) \ln \left( \frac{2p(w|M_A^r)}{p(w|M_A^r) + p(w|M_B^r)} \right) + & \\ + \sum_{w \in \Sigma^k} p(w|M_B^r) \ln \left( \frac{2p(w|M_B^r)}{p(w|M_A^r) + p(w|M_B^r)} \right), \text{else} & \end{cases}$$

Furthermore, Dai *et al.* present a statistical model, denoted by $\xi = (M, W)$, that contains probabilistic information from both Markov chains $M$ and $k$-word distributions $W$. Let $\xi_{A,k}^r = (M_A^r, W_k^A)$ and $\xi_{B,k}^r = (M_B^r, W_k^B)$ be two probabilistic models of $A$ and $B$, respectively. The *weighted relative entropy* between those two probability distributions is defined as

$$wre.k.r(\xi^r_{A,k}, \xi^r_{B,k}) = \sum_{w\in\Sigma^k} \left| \varphi(w|\xi^r_{A,k}) \ln \left( \frac{2\varphi(w|\xi^r_{A,k})}{\varphi(w|\xi^r_{A,k}) + \varphi(w|\xi^r_{B,k})} \right) \right|$$

where, for each $1 \leq k \leq \min\{n, m\}$, $\varphi(w|\xi^r_k)$ denotes the entry corresponding to the $k$-word $w$ under the statistical model $\xi^r_k$. As already outlined, it can be computed by multiplying $p(w|M^r)$ as defined previously by the probability obtained via the frequency of the $k$-word $w$ in the string. Finally, the symmetric form of $wre.k.r$, denoted by $S_2.k.r$, is defined by $S_2.k.r(\xi^r_{A,k}, \xi^r_{B,k})$ as follows:

$$\begin{cases} 0 & \text{if } \xi^r_{A,k} = \xi^r_{B,k} \\ \left( \sum_{w\in\Sigma^k} \varphi(w|\xi^r_{A,k}) \ln \left( \frac{2\varphi(w|\xi^r_{A,k})}{\varphi(w|\xi^r_{A,k}) + \varphi(w|\xi^r_{B,k})} \right) + \right. & \\ \left. + \sum_{w\in\Sigma^k} \varphi(w|\xi^r_{B,k}) \ln \left( \frac{2\varphi(w|\xi^r_{B,k})}{\varphi(w|\xi^r_{A,k}) + \varphi(w|\xi^r_{B,k})} \right) \right) /|\Sigma^k| + 2\ln 2, \text{ else} & \end{cases}$$

and it is proved to be a valid distance measure.

With the use of receiver operating characteristic (ROC) analysis, Dai *et al.* compare the performance of their measures with respect to other existing ones. More precisely, they first compare $wre.k.r$ with some similarity measures based on alignment and $k$-word distributions, such as ClustalW, cosine of the angle, Euclidean distance, Pearson's correlation coefficient, and empirical relative entropy. Then they compare the measures $wre.k.r$ and $S_2.k.r$, based on Markov model plus $k$-word distributions, with some others similarity measures based on Markov model, such as $D_2$ [55] and $D_2z$ [47] (see Section 16.5 for further details on these measures), *SimMM* [67], $rre.k.r$, and $S_1.k.r$. They show that *wre* performs better than other alignment-based or alignment-free methods for similarity searches, whereas its symmetrical form $S_2.k.r$ has no significant improvement.

Because the measure $S_2.k.r$ can be seen as a statistical distance measure, they also use it to construct phylogenetic trees. In this case, they show that the so obtained trees are in good agreement with benchmark phylogenies, which indicates that the distance $S_2.k.r$ is a good measure for phylogenetic analysis.

### 16.2.3 A Method Based on Statistical Dependency, via Mutual Information

In the specialistic literature, there are plenty of alignment-free techniques based on information-theoretic ideas that attempt to quantify the statistical correlation among various parts of biomolecules, such as DNA, RNA, and proteins (see [38] and the references therein). But none of them addresses the problem of how statistically significant the computed dependency is. To the best of our knowledge, Aktulga *et al.* [12] are the first to propose a solution to this problem via a methodology based on the notion of mutual information. Indeed, they define a threshold function that quantifies the level of significance of dependencies between strings. Such a function is connected to the probability of error (*i.e.*, declaring two strings statistically dependent when they are not and vice versa).

Consider again $A$ and $B$ and assume that the two strings come from possibly different independent and identically distributed (iid) probability distributions. Recall that we denote by $n$ the length of $A$ and by $m$ the length of $B$. We assume that $m > n$. Consider now the following two possible scenarios that formalize the notions of dependence and independence:

- *Dependence*: $A$ and $B$ are dependent (*i.e.*, after having generated the sequence $A$, an index $j$, $1 \leq j \leq m - n + 1$, is chosen in an arbitrary way, and a word $B_j^{j+n-1} = b_j b_{j+1} \cdots b_{j+n-1}$ of length $n$ is generated as the output of a discrete memoryless channel having input $A$). The rest of the symbol in $B$ generate i.i.d. according to its probability distribution.
- *Independence*: $A$ and $B$ are independent (*i.e.*, the scenario of the previous point does not hold).

To distinguish between the two scenarios, Aktulga *et al.* propose computing the empirical mutual information between $A$ and each factor of $B$ of length $n$. For each integer $j$, $1 \leq j \leq m - n + 1$, let $\{p_{(i,l)}^{A,B_j}\}$ be the joint empirical distribution of $(A, B_j^{j+n-1})$ (*i.e.*, $p_{(i,l)}^{A,B_j}$ is the proportion of the $n$ positions $(a_1, b_j)$, $(a_2, b_{j+1})$,$\cdots$, $(a_n, b_{j+n-1})$), where $(a_t, b_{j+t-1})$ equals $(s_i, s_l)$, $s_i$ ,$s_l$ in $\Sigma$. Similarly, let $\{p_i^A\}$ and $\{q_l^{B_j}\}$ be the empirical distribution of $A$ and $B_j^{j+n-1}$, respectively. Hence, the empirical mutual information $I_j(n)$ between $A$ and $B_j^{j+n-1}$ is defined by applying the classical definition of mutual information to the empirical distributions that have been just defined as follows:

$$I_j(n) = \sum_{i=1}^{|\Sigma|} \sum_{l=1}^{|\Sigma|} p_{(i,l)}^{A,B_j} \log \frac{p_{(i,l)}^{A,B_j}}{p_i^A q_l^{B_j}}$$

The interesting novelty of the approach proposed by Aktulga *et al.* consists in the use of a statistical test to capture dependence between $A$ and $B$. For this purpose, they fix a threshold $\theta > 0$ and compute the empirical mutual information $I_j(n)$ between $A$ and each factor $B_j^{j+n-1}$ of length $n$ of $B$. So, if $I_j(n) > \theta$ for some $j$, then the sequences $A$ and $B$ are declared dependent; otherwise, they are declared independent. There are two kinds of errors this test can make: declaring that the two strings are dependent when they are not and vice versa. Aktulga *et al.* provide two asymptotic estimates for those two errors reported as follows:

- *Independence*: for a fixed threshold $\theta > 0$ and large $n$, the probability of error is

$$P_{e,1} = \Pr\{\text{declare dependence} \mid \text{independent strings}\} =$$
$$= \Pr\{I(n) > \theta \mid \text{independent strings}\} \approx \exp\{-(\theta \ln 2)n\}$$

- *Dependence*: if $I$ is the true value of the mutual information, then for any fixed threshold $\theta < I$, and for large $n$, the probability of error is

$$P_{e,2} = \Pr\{\text{declare independence} \mid \text{dependent strings}\} =$$
$$= \Pr\{I(n) \leq \theta \mid \text{dependent strings}\} \approx \exp\left\{-\frac{(I-\theta)^2}{2\sigma^2}n\right\}$$

  where $\sigma^2$ is the variance of a given Gaussian distribution (see [12] for details).

For both probabilities of error to decay to zero for large $n$, the threshold $\theta$ needs to be strictly between 0 and $I$. So it is necessary to have some prior information about the value of $I$ that indicates the desired level of dependence. In practice, however, it is unreasonable to expect to be able to specify in advance the exact kind and level of dependence one wishes to detect in the data. Because the probability of error of the first kind $P_{1,e}$ only depends on $\theta$ (at least for large $n$), and because, in practice, declaring false positives is much more undesirable than overlooking potential dependence, in experiments, one can set an acceptably small false-positive probability $\epsilon$ and then, based on it, compute the threshold $\theta$ by setting $P_{e,1} \approx \epsilon$. For instance, $\epsilon = 0.001$.

Aktulga *et al.* present experimental results for the problem of detecting statistical dependency between different parts in a DNA sequence that indicate the appropriateness of their method.

## 16.3 COMBINATORIAL ALIGNMENT-FREE METHODS

This section is devoted to the presentation of some combinatorial alignment-free methods. From the ones available in this class, we have chosen to describe those that either already have been used for large-scale, genome-wide, studies or that have the potential to perform well in those studies. Indeed, as research moves toward "system-wide" views of biology, phylogenetic reconstructions based on entire genomes become increasingly important. Moreover, there are cases in which phylogenetic reconstruction based on traditional methods, as maximum likelihood or maximum parsimony, seem to be of limited applicability. For instance, in the study of viruses in which different families share very few genes.

The first subsection presents a distance that is based on computing the average lengths of maximum common substrings of two strings. The second subsection presents a distance based on an extension to multisets of the well-known Burrows-Wheeler Transform. The last subsection introduces a method that allows classifying strings according to the local similarity of segments of a fixed length $N$ that they share.

### 16.3.1 The Average Common Substring Distance

This distance, introduced by Ulitsky *et al.* [76] and referred to as average common substring (ACS), is deeply related to the Kullback-Leibler distance and has the great potential to scale well to deliver phylogenies involving higher eukaryotic genomes.

Consider strings $A$ and $B$. For any $i$, $1 \leq i \leq n$, let $l(i)$ be the length of the longest substring of $A$ starting at position $i$ that occurs also in $B$ and let

$$L(A, B) = \sum_{i=1}^{n} \frac{l(i)}{n}$$

The ACS distance is defined as

$$\mathrm{ACS}(A, B) = \frac{d(A, B) + d(B, A)}{2}$$

where

$$d(A, B) = \frac{\log m}{L(A, B)} - \frac{\log n}{L(A, A)}$$

Ulitsky *et al.* use suffix arrays [41] to compute ACS efficiently, which turns out to be efficient enough to support large-scale experiments. Nevertheless, recall that a suffix array [41] is an array of integers giving the starting positions of suffixes of a string in lexicographical order. The fundamental and most expensive step for its construction, in particular in terms of main memory, is the sorting of the suffixes. Although there has been quite a bit of investigation on space-conscious indexing data structures [31,59], none of the available techniques seem to scale well with sequence lengths to grant phylogenetic reconstruction for genomes in the gigabases.

It can be shown [76] that ACS is related to the relative compressibility of two Markov induced distributions. As argued by Ulitsky *et al.*, if $A$ and $B$ are two strings generated by a pair of Markovian distributions $p$ and $q$, then $d(A, B)$ converges to $\tilde{\mathrm{D}}(q \| p) = -E_p(\log(q(X))$, as the length of $A$ and $B$ goes to infinity.

To assess the performance of ACS, the authors compare it with some reference methods [52,61,69] on benchmark datasets, reporting satisfactory results. Moreover, they go a step further and show that ACS can be used to generate the genome phylogenomic forest for almost 2000 viruses and the proteome phylogenomic tree for hundreds of species. For details on experiments, see Section 16.6.

### 16.3.2 A Method Based on the EBWT Transform

Mantaci *et al.* [57] present a methodology that provides one of the first applications of the well-known Burrows and Wheeler transform (BWT) [19] outside the realm of data compression. Indeed, the authors define a new distance measure that is based on an extension to multisets of the BWT. We first will describe the extended Burrows–wheeler transform (EBWT) transform and the class of distances based on it. Given two strings $A$ and $B$, we say that

$$A \preceq_\omega B \Leftrightarrow A^\omega <_{\text{lex}} B^\omega$$

where the relation $<_{\text{lex}}$ denotes the well-known lexicographic order and, for any $x \in \Sigma^*$, $x^\omega$ is defined as $x^\omega = xxxx \ldots$. Although the $\preceq_\omega$ order is defined on infinite words, by using the Fine and Wilf Theorem [33], Mantaci *et al.* show that one can decide the mutual $\preceq_\omega$ ordering between $A$ and $B$ in linear time because only their lengths up to $|A| + |B| - \gcd(|A|, |B|)$ matter.

Now fix two colors, $R$ for red and $W$ for white. The application $\gamma$ defined by $\gamma(A) = R$, $\gamma(B) = W$ is the *coloring* of $(A, B)$. Recalled that two words $u$ and $v$ in $\Sigma^*$ are conjugates if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$, it is possible to extend the coloring $\gamma$ to the set of all conjugates of $A$ and $B$, $\text{Conj}(A, B)$, as follows: $\forall C \in \text{Conj}(A, B)$

$$\gamma(C) = \begin{cases} R & \text{if } C \text{ has been obtained as conjugate from } A \\ W & \text{if } C \text{ has been obtained as conjugate from } B \end{cases}$$

Let $M$ be the matrix with three columns and $|A| + |B|$ rows in which for every $C \in \text{Conj}(A, B)$, each row is of the form $(C, L(C), \gamma(C))$, where $L(C)$ denotes the last letter of word $C$. Sort, now, rows in $M$ by taking as sorting key the first component of each row, using the $\preceq_\omega$ order, and second sorting key the third component of the triplets, $\gamma(C)$, by considering $R < W$. The second and third columns of $M$, $M_L$ and $M_\gamma$ is what Mantaci *et al.* denote the $\gamma$ or $\text{EBWT}(A, B)$. Table 16.1 presents an example.

The colored EBWT is used in [57] to define a class of distance measures between strings. In fact, if $A$ and $B$ are two primitive strings and $\mathcal{P}$ is a parsing of $M_\gamma$, then the *distance of $A$ and $B$ associated to the parsing* $\mathcal{P}$ is:

$$D_{\mathcal{P}}(A, B) = \sum_{x \in \mathcal{P}} |n_R^x - n_W^x|$$

where $n_R^x$ (or $n_W^x$) counts the number of characters colored by $R$ (or $W$) in the factor $x$ of $M_\gamma$, $x$'s being the blocks of the parsing.

Notice that a distance measure exists for each parsing of $M_\gamma$. These measures are symmetric, but the property of identity of indiscernible does not age. In fact, if

**Table 16.1 Table representing the matrix $M$ corresponding to the $\gamma$-EBWT($u$, $v$), where $u = abaababba$ and $v = baabbaba$.**

| $i$ | MC | $M_L$ | $M_\lambda$ |
|---|---|---|---|
| 1 | aababbab | $b$ | $R$ |
| 2 | aabbabab | $b$ | $W$ |
| 3 | abaababb | $b$ | $R$ |
| 4 | abaabbab | $b$ | $W$ |
| 5 | ababaabb | $b$ | $W$ |
| 6 | ababbaba | $a$ | $R$ |
| 7 | abbabaab | $b$ | $R$ |
| 8 | abbababa | $a$ | $W$ |
| 9 | baababba | $a$ | $R$ |
| 10 | baabbaba | $a$ | $W$ |
| 11 | babaabab | $b$ | $R$ |
| 12 | babaabba | $a$ | $W$ |
| 13 | bababaab | $b$ | $W$ |
| 14 | babbabaa | $a$ | $R$ |
| 15 | bbabaaba | $a$ | $R$ |
| 16 | bbababaa | $a$ | $W$ |

$A = B$, then it is sufficient to choose a parsing with blocks of odd length to obtain $D_{\mathcal{P}}(A, B) > 0$.

An important property proved in [57] is that a connection exists between the distance defined by these authors and the $k$-tuple count Euclidean distance. Indeed, for any positive integer $k \leq \min\{|A|, |B|\}$, it is possible to find a parsing $\mathcal{P}(k)$ of $M_\gamma(A, B)$, depending on $k$, such that $D_{\mathcal{P}(k)}$ approximates $D_k$. In practice, the authors give evidence that it is sufficient to choose a relatively "small" $k$ (*i.e.*, $k = 10$), to obtain a good approximation.

Mantaci *et al.* also introduce another distance that represents a semimetric, being symmetric and preserving the identity of indiscernible, but it not a metric because the triangle inequality does not hold. This new distance, called the *monotonic block distance* and denoted by $D_{BW}$, is defined as the distance of $A$ and $B$ associated to the monotonic block parsing $\mathcal{M}$ (*i.e.*, to the parsing that decomposes the column $M_L$ of EBWT in blocks made each by equal characters).

Experiments in [57] prove that this new distance is a very good measure for mitochondrial genome phylogeny. Note that, as is stressed for the ACS distance, also here the suffix sorting step is a computational bottleneck.

### 16.3.3 *N*-Local Decoding

Although most alignment-free methods known in the literature can be seen as a global synopsis of two strings, Didier *et al.* [29] introduce a method that captures local similarity of sequences. These definitions are deeply investigated in [24] in which the authors stress the importance of the method.

We first introduce the notion of local decoding and then the dissimilarity functions one can define on that notion.

Let $S = \{A, B, C, \ldots\}$ be a set of sequences over the alphabet $\Sigma$. The $j$th *site* of any string $A$ in $S$, $1 \leq j \leq |A|$, is defined as the pair $\sigma = (A, j)$. By convention, $\sigma + c = (A, j + c)$, for $1 \leq j + c \leq |A|$.

Given a positive integer $N$, the *$N$-neighborhood* of site $\sigma = (A, j)$ is the set of sites $(A, m_N) \ldots (A, M_N)$, where $m_N = \sup\{1, j - N + 1\}$ and $M_N = \inf\{|A|, j + N - 1\}$ (*i.e.*, the substring of $A$ of length $2N - 1$ centered in $j$ and possibly truncated at the ends of $A$). A string $B$ of length $N$ is *in relative position $l$ with respect to the site $\sigma = (A, j)$* if $B$ matches exactly the substring of $A$ beginning at position $j - l$.

Two sites $\sigma$ and $\sigma'$ are said to be *directly related*, $\sigma \simeq_N \sigma'$, if and only if a string $B$ of length $N$ exists at the same relative position in the $N$-neighborhoods of $\sigma$ and $\sigma'$. This implies that $\sigma$ and $\sigma'$ have to be occupied by the same letter. Two sites $\sigma$ and $\sigma'$ may be related directly to a third one $\sigma''$, even if they are not directly related. This depends on the relative positions in the $N$-neighborhoods of the sites in which their associated strings coincide. Anyway, $\sigma$, $\sigma'$, and $\sigma''$ will be occupied by the same letter.

It is natural to extend this definition as follows. Two sites $\sigma$ and $\sigma'$ are *related* $\sigma \sim_N \sigma'$ if there is a chain of direct relations that links them. This transitive closure represents an equivalence relation among the set of sites of $S$ and induces on it a partition, called the *$N$-local decoding of $S$*.

It is convenient to recall that a suffix tree data structure is a trie in which all substrings of a given string are stored. An example is given in Figure 16.1. Its construction takes linear time, and it can be stored in linear space. The interested reader can find an excellent introduction to suffix trees and its applications in [41].

Didier *et al.* also present an algorithm for computing the $N$-local decoding of a given set $S$. Their construction relies on that of the suffix tree associated to the string obtained by catenating sequences of $S$ through special symbols not belonging to the alphabet $\Sigma$. Leaves in that suffix tree are indexed by the sites of $S$ (and the external symbols). A node of depth $N$ is a common ancestor of two leaves $\sigma$ and $\sigma_0$ if they are the starting site of a common word of length $N$ (*i.e.*, if $\sigma \simeq_N \sigma'$). Therefore, the

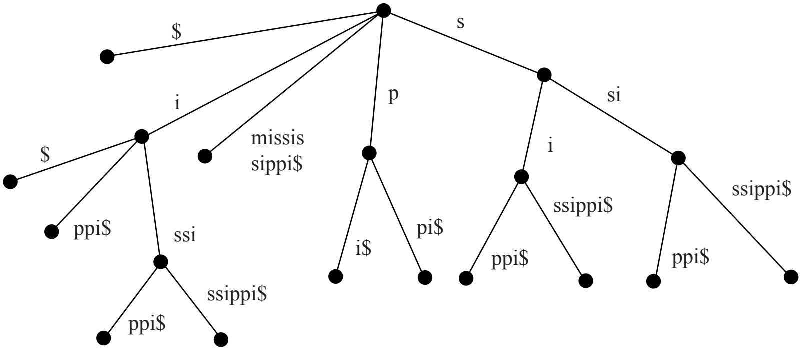


**Figure 16.1** Suffix tree of word *mississippi*\$.

described suffix tree yields a very fast procedure for the computation of the $N$-local decoding.

Based on $N$-local decoding, it is possible to define two dissimilarity scores. Let $A$ and $B$ be two strings, $s_i$ be any character in both $A$ and $B$, and $n_i^A$ (or. $n_i^B$) be the number of occurrences of $s_i$ in $A$ (or $B$). The similarity between $A$ and $B$ is given by the quantity

$$\mathrm{sim}(A, B) = \frac{\sum_{s_i} \min(n_i^A, n_i^B)}{\min(|A|, |B|)}$$

that is, the ratio between the smallest number of occurrences of $s_i$ in the two sequences and the length of the shorter sequence.

Now, let $\{A^N, B^N\}$ represent the $N$-local decoding of $\{A, B\}$, the *$N$-local dissimilarity score* between $A$ and $B$ is given by

$$\mathrm{dist}L_N(A, B) = 1 - \mathrm{sim}(A^N, B^N)$$

Moreover, by denoting with $A[N]$ (or $B[N]$) the sequence of successive overlapping strings of length $N$ of $A$ (or $B$), it is possible to define another distance, the *$N$-block dissimilarity score between $A$ and $B$*, as

$$\mathrm{dist}B_N(A, B) = 1 - \mathrm{sim}(A[N], BN])$$

Didier *et al.* evaluate the accuracy of these two dissimilarities over several datasets. They obtain essentially the same results with respect to more realistic alignment methods. Moreover, they compare their results, in the case of $N = 10$, with the ones obtained with another alignment-free method, the one by Pham and Zuegg [67]. This last dissimilarity, based on short words, shows lower correlation with the reference than $\mathrm{dist}L_{10}$ and $\mathrm{dist}B_{10}$. $N$-local decoding also has been applied to construct trees for the subtyping of Human Immunodeficiency Virus (HIV) and Simian Immunodeficiency Virus (SIV) variants [28].

## 16.4 ALIGNMENT-FREE COMPOSITIONAL METHODS

In this section we describe some alignment-free methods based on the notion of subword composition of sequences, which, thanks to extensive experimentation, have become significant for biological data mining. The first subsection is devoted to illustrate the $k$-string composition approach, as described in [69]. The second subsection is devoted to a generalization of that method introduced in [86]. It also briefly describes another variant [56] that is supposedly more robust and efficient in performing sequence comparison with respect to the previous methods. The third subsection covers algorithmic issues relating to the efficient implementation of the method described in Section 16.4.1.

### 16.4.1 The *k*-String Composition Approach

Qi *et al.* [69] propose a method to infer evolutionary relatedness of microbial organisms that is based on the $k$-word frequencies taken over the alphabet of amino acids. They also show that the method can applied be successfully to phylogenetic studies of entire proteomes. Although analogous methods already had been proposed with some success in the literature, the novelty of their approach is to account for background letter probability distributions to factor out the amount of information in a sequence caused by "evolutionary pressure" as opposed to random processes. Moreover, the choice of the domain of application is also well motivated. Indeed, the authors propose determining an evolutionary distance between two organisms by counting the olipeptide strings of a fixed length $k$ in the collection of their protein sequences. They observe that the mutation rates are higher when one considers noncoding segments in the genomes. So, translated amino acid sequences from coding regions of DNA are considered more significant to obtain phylogenetic relations. We now provide a formal definition of their method.

Consider $A$ and a $k$-word $\alpha = \alpha_1\alpha_2 \ldots \alpha_k$. The probability of occurrence of $\alpha$ in $A$ is

$$p(\alpha) = \frac{f(\alpha)}{(n-k+1)}$$

where $f$ denotes the number of occurrences of $\alpha$ in $A$.

When $A$ is a biological sequence, the probability distribution induced by $p$ over $\Sigma^k$ accounts for random mutations as well as effective evolution. To emphasize the selective diversification during evolution rather than the random mutations, the authors propose to subtract a random background from $p(\alpha)$. That is done by considering the probability of appearance of a $k$-word in terms of $(k-1)$-words, where the probability of those latter are computed via a Markovian background model of order $k-1$. That is,

$$p^0(\alpha) = \frac{p(\alpha_1\alpha_2 \ldots \alpha_{k-1})p(\alpha_2\alpha_3 \ldots \alpha_k)}{p(\alpha_2\alpha_3 \ldots \alpha_{k-1})}$$

Notice that the probability of a $k$-word now depends on the probability of words of length $k-1$ and $k-2$, *as generated* by the background Markov model.

The difference between $p$ and $p^0$ gives the real information about the evolutionary process. Now, for each $k$-word $\alpha$, let

$$F_A(\alpha) = \begin{cases} \dfrac{p(\alpha) - p^0(\alpha)}{p^0(\alpha)} & \text{if } p^0 \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Next, fix an arbitrary order of the words in $\Sigma^k$ and refer to the $i$th word in it simply as $i$. Finally, the composition vector (CV) for $A$ is $(F_A(1), F_A(2), \ldots, F_A(N))$, where $N = |\Sigma|^k$.

Given two strings $A$ and $B$ and the respective CVs $(F_A(1), F_A(2), \ldots, F_A(N))$ and $(F_B(1), F_B(2), \ldots, F_B(N))$, the correlation between $A$ and $B$ is defined as follows:

$$C(A, B) = \frac{\sum_{i=1}^{N} F_A(i) \times F_B(i)}{(\sum_{i=1}^{N} F_A(i)^2 \times \sum_{i=1}^{N} F_B(i)^2)^{\frac{1}{2}}} \tag{16.1}$$

Because $C$ varies between $-1$ and 1, the distance between $A$ and $B$ is defined as follows:

$$D(A, B) = \frac{1 - C(A, B)}{2}$$

Qi *et al.* have used such a distance to build phylogenetic trees of 109 organisms. Note that $k$ is a parameter of the method, and in fact Qi *et al.* have studied how the topology of the phylogenetic tree they build varies with $k$. Remarkably, their experiments show that the topology of the tree exhibits less and less variation as $k$ increases. Moreover, for the taxa they examined, such a topology becomes stable for $k = 5$ and 6.

### 16.4.2 Complete Composition Vector

This method, introduced in [86], integrates the strategy described in the previous subsection and the notion of complete information set. The latter was proposed by Li *et al.* [54] and consists of considering the occurrence probability $p(\alpha_1\alpha_2 \ldots \alpha_k)$ for each $k$-word and for each $k$, $1 \leq k \leq n$. Each of those probability distributions gives the $k$th information set $U^k$ for the sequence $A$. The collection of all information sets $(U^1, U^2, \ldots, U^n)$ is referred to as the complete information set of $A$.

The introduction of the complete composition vector (CCV for short) is motivated by a problem in the CV. Indeed, the subtraction stage disconnects the $k$th composition vector and the $k - 1$th one. The approach here provides the lost information by using a collection of CVs $(U^{k_1}, U^{k_1+1}, \ldots, U^{k_2})$, where $k_1 \leq k_2$ are two *a priori* fixed bounds on the word sizes. Those values are determined experimentally, and one appropriate setting seems to be $k_1 = 3$ and $k_2 = 7$.

Given a word size range $[k_1, k_2]$, for two strings $A$ and $B$, the respective CCVs are $(A^{k_1}, A^{k_1+1}, \ldots, A^{k_2})$ and $(B^{k_1}, B^{k_1+1}, \ldots, B^{k_2})$. The correlation $C(A, B)$ between $A$ and $B$ is defined as follows:

$$C(A, B) = \frac{\sum_{j=k_1}^{k_2} \sum_{i=1}^{N} f_i^j \times g_i^j}{(\sum_{j=k_1}^{k_2} \sum_{i=1}^{N} (f_i^j)^2 \times \sum_{j=k_1}^{k_2} \sum_{i=1}^{N} (g_i^j)^2)^{\frac{1}{2}}}$$

where $f_i^j$ and $g_i^j$ are the $i$th entry in the $j$th CVs of $A$ and $B$, respectively. The distance between $A$ and $B$ is defined as follows:

$$D(A, B) = -\ln\left(\frac{1 + C(A, B)}{2}\right)$$

The method has been applied to infer evolutionary information for several biological datasets.

Although the CCV distance gives finer evolutionary information than its CV counterpart, it requires nontrivial computational resources (*i.e.*, both time and memory). To address that problem, an improved version of the CCV distance is provided in [56]. The main improvement consists of constructing the CCVs by using the frequencies of all $k$-words normalized via expectation and variance:

$$\frac{f(\alpha) - E[f(\alpha)]}{\sqrt{\mathrm{Var}[f(\alpha)]}}$$

It is somewhat unfortunate that such a suggested speed-up has very serious drawbacks. In fact, the computation of expectation and variance are given for the case in which the background model from which a random string is generated is iid. This is a serious limitation because there is no account of the so-called "context dependencies" within a string. Such a shortcoming can be solved with the use of a background Markov model, but in that case, the computation of expectation and variance are highly nontrivial tasks, both mathematically and computationally, as briefly discussed in Section 16.5.1.

### 16.4.3 Fast Algorithms to Compute Composition Vectors

A straightforward computation of distances based on CVs takes time exponential in $k$, severely limiting their application range. It is also fortunate, as noticed and exploited by Apostolico and Denas [16], that those distances can be computed in time linear in the length of the input sequences via the use of suffix tree data structures. Apostolico and Denas compute a generalization of the CV distance in which all $k$-words, up to a fixed length $K$, contribute to the distance. Because the algorithm is a nearly standard application of suffix trees, we limit ourselves to mention its main points.

A variant of a suffix tree is built in which all strings of length up to $K$ are considered. This means that the tree is truncated at depth $K$. Intuitively, the computation of the distance between two sequences $A$ and $B$ is done by considering their respective truncated suffix trees, drawn in different colors and then superimposed. Note that only the strings occurring in both $A$ and $B$ contribute to the numerator of the distance formula (see Formula (16.1)). Those words are exactly the ones that can be found on paths and nodes bearing both colors. The words found on a path with only one color contribute to only one sum appearing in the denominator (see Formula (16.1)). There are also some words that do not appear in the sequences but could contribute to the values of the composition vectors because their substrings of length $k - 1$ and $k - 2$ appear in either or both of $A$ and $B$. Such words are referred to as *chimeral words* in [16]. They are words of the form avb, where avb does not appear in the considered sequence but both av and vb do. It is interesting to notice that chimeral words correspond to a well-known combinatorial notion known as *minimal forbidden words* [58]. Besides the notion of chimeral words, also the notion of maximal and nonmaximal word are of use. In particular, a word is maximal when it is impossible to extend it by a character without losing some of its occurrences. The contribution of such words to the distance between two sequences can be computed in linear time. Indeed, a word is maximal if and only if it ends at a node of the truncated suffix tree. Its frequency count can be computed in constant time, via a bottom-up visit of the tree. Therefore, the contributions of maximal words to the composition vectors can be computed in overall linear time. The case of nonmaximal words can be handled without increasing that time complexity. Indeed, a nonmaximal word is a word terminating in the middle of an arc of the suffix tree, and therefore, it has the same frequency count as its shortest extension that is a maximal word. Also the case of chimeral words is considered, and also in this case, the computation of the contribution given by those words to the distance formula can be done in linear time.

## 16.5 ALIGNMENT-FREE EXACT WORD MATCHES METHODS

Among the plethora of alignment-free methods that have been developed, the one nicknamed $D_2$ stands out for its mathematical elegance as well as its usefulness. Indeed, it is among the few measures that lends itself to rigorous mathematical studies, and it is a standard for Expressed Sequence Tag (EST) clustering and database searches as well as *ab initio* discovery of *cis-regulatory modules*. The interested reader can find references to both its introduction and its initial uses in [55], whereas this section focuses on an outline of its main statistical properties and relevant extensions. The remainder of this section is organized as follows. The first subsection formally defines $D_2$ and presents some of its statistical properties useful for database searches. Subsection 16.5.2 presents a generalization of $D_2$ as well as a procedure to determine experimentally the best word size $k$ for its use. Although that procedure has been proposed for $D_2$, it is fully general and can be useful for other methods based on $k$-words. Finally, Subsection 16.5.3 contains a variant of $D_2$, referred to as $D_2z$, which offers several advantages with respect to $D_2$. This new measure is a *de facto* z-score indicating how many standard deviation units the computed value of $D_2$ is away from its expected value.

### 16.5.1 $D_2$ and its Distributional Regimes

The $D_2$ measure is defined as the number of $k$-word matches between the two sequences $A$ and $B$, including overlaps. Formally it is expressed as follows:

$$D_2(A, B) = \Sigma_{(i,j)\in I} Y_{(i,j)} \tag{16.2}$$

where $Y_{(i,j)}$ is an indicator variable equal to one if and only if the substring of length $k$ starting in position $i$ of $A$ is equal to the one starting in position $j$ of $B$, $I = \{(i, j) : 1 \leq i \leq \overline{n}$ and $1 \leq j \leq \overline{m}\}$, with $\overline{n} = n - k + 1$ and $\overline{m} = m - k + 1$. $D_2$ can be computed in linear time, with the use of a suffix tree [41], for any word size. Let $N^A$ be the word count vector of $A$. It is obtained by listing, in lexicographic order, the frequency counts $f(\alpha)$ of all $k$-words in $\Sigma^k$. $N^B$ is defined analogously. Then, it can be shown easily that $D_2(A, B)$ is equal to the inner product of those two vectors.

For any similarity/distance function $F$ between two strings to be a valuable tool for biological data mining and, in particular, database searches, it is essential to have associated it with methodologies that establish how "surprising" or "unusual" the value $F(A, B)$ is (*i.e.*, whether it is significantly different than what one would expect if $A$ and $B$ were correlated randomly). A good and well-known example is given by the BLAST program in which the $e$ value associated to an alignment indicates how likely it is for that alignment to have occurred by chance. Unfortunately, those methodologies are rare because of the mathematical difficulties associated with the task of deriving them. In any case, a first step toward that goal, which for $D_2$ is still open, is to identify the distributional regime of $F$ (*i.e.*, when $F$ is considered as a random variable one would be interested to know, for instance, whether $F$ behaves according to a Poisson or a normal distribution. In regard to $D_2$, investigation into those issues has been initiated in [55], with additional results presented in [45]. A summary follows.

Assume that the two sequences have been obtained by means of an iid background model. Then, the following results hold:

(a) When the letter distribution is not uniform (*i.e.*, all letters are not equiprobable, and $k > 2\log_b n$), the distribution of $D_2$ has a compound Poisson asymptotic behavior, where $b$ is a rather complex function of the letter probabilities (see [55] for details).

(b) Again, when the letter distribution is not uniform and $k < 1/6 \log_b n$, the distribution of $D_2$ has a normal asymptotic behavior.

(c) When the letter distribution is uniform and $k = \alpha \log_b n + C$, $0 < \alpha < 2$ and $C$ constant, the distribution of $D_2$ has a normal asymptotic behavior. We point out, however, that there are example from Lippert *et al.* [55] showing that for uniform letter distributions, the distribution of $D_2$ is neither Poisson nor normal.

Apart from the technical merit of the results just outlined, they also have a great practical significance. Indeed, they indicate that as the word size changes, the behavior of the random variable $D_2$ changes, implying that significance tests on its value must take into account the word size. It is unfortunate that no full characterization of those distributional changes, as a function of $k$ is yet available for $D_2$. However, there are experiments indicating that the "boundary" between the Poisson and the normal distributional regimes (*i.e.*, (a) and (b)), is close to $k = 2\log_b n$ [46].

### 16.5.2 An Extension to Mismatches and the Choice of the Optimal Word Size

A straightforward extension of $D_2$ to approximate $k$-word matches has been proposed in [35]. Fix an integer $t \geq 0$ and let

$$D_2^t(A, B) = \Sigma_{(i,j)\in I} Y_{(i,j)} \tag{16.3}$$

where the notation is as in Equation (16.2), except that $Y(i, j)$ is equal to one if and only if the two substrings involved in the comparison have a Hamming distance of at most $t$. The distributional regimes of $D_2^t$ have been studied in [35, 46]. In terms of theoretic results, state of the art is far from an even partial characterization of the distributional regimes of $D_2^t$ along the same lines of points (a)–(c) of the previous subsection. However, with the use of Kolmogorv–Smirnov tests, experiments show that $D_2^t$ is in good agreement with the normal distributions for various word sizes.

Foret *et al.* [35], in their investigation of the mathematical properties of $D_2^t$, also proposed a methodology to estimate experimentally the optimal word size for the use of $D_2^t$. That is, a method to estimate the value of $k$ that would capture best the similarity among sequences when evaluated by means of $D_2^t$. Although hardly discussed, such an estimation is a key step for most measures presented in this chapter. As we bring to light here, it is fortunate that the method proposed by Foret *et al.* for $D_2^t$ is general, elegant, and particularly simple. The following procedure presents it in terms of a generic similarity function $F_k$, where $k$ is a parameter of the function indicating the word size it uses. It takes as input $F$, a "seed" string $A_0$ of length $n$ and two integers $g > 0$ and $k_{\max}$, whose role is specified in the procedure. It returns $K_{\text{opt}}$, a (possibly singleton) set of word sizes for which $F$ is "most discriminative."

**Algorithm 16.1 K-OPT** $(F, A_0, g, k_{\max}, K_{\text{opt}})$

1. **Generate a gold standard dataset**. Generate an increasing sequence of integers $\gamma_i$, $1 \leq i \leq g$, in the range [1, 100]. Generate a set of sequences $A_i$, $1 \leq i \leq g$, such that each $A_i$ is at Hamming distance $\lfloor \frac{n\gamma_i}{100} \rfloor$ from $A_0$, (*i.e.*, $A_0$ and $A_i$ "differ" by $\gamma_i\%$). Let $\text{RANK}_0$ be a vector of length $g$ such that $\text{RANK}_0[i] = i$ (*i.e.*, it is the ranking of the strings $A_i$ according to their dissimilarity from $A_0$).
2. **Compute dissimilarity rank vectors using** $F$. For $1 \leq k \leq k_{\max}$, compute $F_k(A_0, A_i)$, $1 \leq i \leq g$, and sort that sequence in decreasing order. Store in $\text{RANK}_k$, the permutation of $1, 2, \cdots, g$ so obtained.
3. **Estimate** $k_{\text{opt}}$. For $1 \leq k \leq k_{\max}$, perform a Spearman rank correlation test between $\text{RANK}_0$ and $\text{RANK}_k$. Return as $K_{\text{opt}}$ the set of values of $k$ for which the Spearman correlation with $\text{RANK}_0$ is maximum.

A few remarks are in order. The seed sequence $A_0$ either may be generated at random, or it may be chosen as a good "representative" of a particular application domain (*e.g.*, coding regions). The value of $g$ must be "large enough" so that the

Spearman correlation test is reliable (small $p$-value). That is usually achieved for $g$ in the hundreds. Hamming distance can be substituted with any other kind of distance to generate the gold standard dataset. For instance, for DNA sequences, Foret *et al.* use the Kimura model of evolution [49] to produce a gold standard dataset from the seed sequence of Human chromosome one. In practice, Algorithm 16.1 is really to be used as a Monte Carlo sampling strategy in which, for various seeds and sequence length, one tries to obtain a consensus set, possibly a singleton, $K_{\text{opt}}$.

The results reported by Foret *et al.* regarding $D_2^t$ show that $K_{\text{opt}}$ is remarkably stable across sequence length and $t$. For instance, a value of $k = 7$ seems to be the best choice for $t = 0$ and sequence length in the range [400, 3000].

### 16.5.3 The Transformation of $D_2$ into a Method Assessing the Statistical Significance of Sequence Similarity

Despite the efforts to characterize the statistical properties of $D_2$, the state of the art gives only a partial picture with some serious limitations. For instance, all results mentioned in the previous two subsections apply only to the iid case in which both sequences are derived from the same memoryless probability distribution. Moreover, there is no measure of how significant a computed value of $D_2$ is.

To address those shortcomings, Kantorovitz *et al.* [47] have introduced a variant of $D_2$, referred to as $D_2z$, which offers several advantages with respect to $D_2$. First, for significance analysis purposes, it is not necessary to assume that the two strings involved in the comparison come from the same iid probability distribution. Indeed, one can assume that the background models associated with the strings are two *different* Markov chains of finite order. Moreover, the new measure is a *de facto* z-score indicating how many standard deviation units the computed value of $D_2$ is away from its expected value. Therefore, very low or very high values of $D_2z$ indicate that the similarity of the two strings involved in the comparison is "surprising." Formally:

$$D_2z(A, B) = \frac{D_2(A, B) - E[D_2]}{\sigma[D_2]} \tag{16.4}$$

where $E[D_2]$ and $\sigma[D_2]$ are the mean and standard deviation of $D_2$ with respect to two background probability models generating $A$ and $B$, respectively.

For the $D_2z$ measures to have the formal rigor of a z-score, one needs to choose background probability models so that $D_2$ is distributed normally. For iid background models, results outlined in Section 16.5.1 are useful here, although they do not cover the entire range of word sizes. When the background models are Markov chains of finite order, Kantorovitz *et al.* resorted to numeric simulations to show that even when the length of the sequences is as short as 400, $D_2$ approximates well the normal distribution, even for finite-order Markov chains.

Another relevant aspect concerning $D_2z$ is the computation of its mean and variance. For iid, those formulas are easy to derive and lend themselves to efficient computation via algorithms with time complexity quadratic in $k$. As for Markov background models, both quantities can be computed in $O(|\Sigma|^k)$ time, although the derivation of those algorithms involves some very sophisticated mathematical techniques related to the computation of the expectation and variance of random strings. We provide an outline next for the expectation only, limiting ourselves to mention that the efficient algorithm for the computation of the variance makes key use of deep approximation results concerning an infinite series of stochastic matrices because of Kleffe and Borodovsky [50].

Let $M_A$ be a Markov chain of order one from which $A$ is generated. Let $p_j^A(c)$ be the probability that $c$ occurs at position $j$ of $A$. As justified in [50], one can assume that $p_j^A(c)$ is independent of $j$ (*i.e.*, one can assume that $M_A$ has a *character stationary probability distribution* $p^{M_A}$). Now, for any given word $\alpha$, its probability $p^A(\alpha)$ of occurrence in $A$ can be written as $p^{M_A}(\alpha_1)p_*^{M_A}(\alpha)$, where $p_*^{M_A}(\alpha)$ is the same occurrence probability but conditioned on the first symbol being alpha$_1$. Such a probability can be computed from the state transition probability matrix of $M^A$. Using the same notation for $B$ and linearity of expectation, the problem of computing $E(D_2)$ can be reduced to that of computing $E[Y_{i,j}]$, as follows:

$$E[Y_{i,j}] = Pr(Y_{i,j} = 1) = \Sigma_{\alpha \in \Sigma^k} p^{M_A}(\alpha_1)p_*^{M_A}(\alpha)p^{M_B}(\alpha_1)p_*^{M_B}(\alpha) \qquad (16.5)$$

One last remark is in order. For each string, its background Markov model is learned from the string itself. That is done via maximum likelihood estimates of the state transition probability matrix. Such an approach of deriving “locally” the background model for a string offers some advantages over a single model for when the family of strings one needs to model does not have homogeneous statistical properties, as in the case of metazoan genomes that have great local variability in base composition.

## 16.6 DOMAINS OF BIOLOGICAL APPLICATION

In this section, we briefly present the computational biology domains in which the distances previously described have been applied. In fact, we believe that to evaluate the effectiveness of such measures for large-scale biological investigations, it is important to show the biological contexts in which they have been experimentally validated. Most biological datasets used in the experiments comprise proteomic or genomic sequences, mitochondrial genomes and proteins, as summarized in the following:

- Phylogeny: information theoretic and combinatorial methods [26,28,57,76,80]
- Phylogeny: compositional methods [16,69,86]
- CIS regulatory modules [26,47]
- DNA sequence dependencies [12]

In the following subsections, we describe the main experimental contributions of the cited papers.

### 16.6.1 Phylogeny: Information Theoretic and Combinatorial Methods

Phylogeny via alignment-free methods has been the object of considerable investigation (*e.g.*, [15, 20, 22, 26, 28, 30, 52, 53, 57, 61, 69, 76, 80]). Among those methods, here we concentrate on the ones we have examined in this chapter [26, 28, 57, 76, 80] and that present phylogenetic trees generated by using either combinatorial or information theoretic distances. For the convenience of reader is we group experiments according to the nature of the datasets that have been used.

In the first part of the experimental assessment of the ACS measure, Ulitsky *et al.* [76] use the same datasets of proteomic and genomic sequences of [61, 69]. The datasets consist of sequences coming from 75 species and the reference phylogenetic tree (the gold standard) is taken to be a maximum likelihood tree based on the small ribosomal subunit rRNAs [23]. The Robinson–Foulds measure, a statistical standard that indicates the "distance" between two trees, is used for evaluation. Ulitsky *et al.* report that the trees obtained with the use of ACS are closer to the reference tree than the ones obtained by other methods, in particular, the ones in [61,69]. It is interesting to report that, for genomic sequences, the improvement of ACS compared with [69] is only about 2%, whereas for proteomes, it is about 17%.

Mitochondrial genomes have been used extensively to assess the quality of phylogenetic methods. In particular, Ulitsky *et al.* use ACS on a set of mitochondrial genomes of 34 mammalian species, which is the one previously used by Li *et al.* [52]. The results between the two methods are comparable but ACS is faster than the method by Li *et al.*

Dai *et al.* extract a dataset of 29 mammalian species, of which five are rodents, from the mithocondrial one studied in [52,61]. The phylogenetic tree constructed by using their measure $S_2.k.r$ is consistent with biological knowledge and state of the art [20, 52, 61]; in particular, marsupials and monotremes are grouped close to each other as well as the five rodents. Three nonmurid rodents (squirrel, dormouse, and guinea pig) are separated from murid rodents, whereas primates are related closely to each other, and finally, ferungulates are grouped in a branch and clustered with primates. Moreover, their method reconfirms the hypothesis of the grouping (rodents, (primates, ferungulates)). These results confirm that $S_2.k.r$ can be considered another efficient distance measure for mithocondrial phylogenetic analysis.

Finally, Mantaci *et al.* [57] also present experiments on phylogenies of mitochondrial genome. In particular, they use the mtDNA genomes of 20 mammals from GenBank. The results they obtain are very close to the ones derived with other approaches [20, 52, 53, 61]. Also in this case, the resulting phylogeny confirms the hypothesis of grouping of the placental mammals (rodents, (primates, ferungulates)).

Most methods described in this chapter become of very little use when the datasets become large. A remarkable result by Ulitsky *et al.* is to show that ACS is a substantial step forward for large phylogenetic studies. Indeed, they show that it is possible to generate a reliable phylogenetic forest for a large dataset of viral genomes. In particular, they use 1865 viral genomes, whose superfamily is known for 1837 of them. A study of the classification so obtained shows that their phylogenetic reconstruction is mostly in agreement with the accepted taxonomy.

Two other phylogenetic studies involving viruses are also worth mentioning. Wang and Zheng [80] select 25 complete virus genomes, 12 coronaviruses from the three isolated typical groups, 12 SARSCoV (severe acute respiratory syndrome coronavirus) strains and a torovirus. They construct phylogenetic trees using weighted sequence entropy distances ($d_1$, $d_2$, $d_3$) as well as some other classical distance measures (*i.e.*, euclidean distance, the linear correlation coefficient, the cosine function, and the information-based similarity index). Their experiments show that their distances are not inferior to those classical methods. Didier *et al.* [28] evaluate their method on HIV and SIV. The tree topologies they obtain agree with those obtained by a combination of standard methods present in the HIV Sequence Compendium.

Finally, Ulitsky *et al.* present a phylogenetic tree based on all existing proteomes in the NCBI database, release 2006. The dataset that the authors consider consisted of 19 proteomes of Archea, 161 proteomes of Bacteria, and 11 proteomes of Eukaryota. The ACS method leads to a tree in which the species are split up correctly in the three main domains, except for two archean species that do not belong to their domain branch. The method accuracy is good if considering genera, families, and classes, whereas it decreases for higher taxonomyc groups.

### 16.6.2 Phylogeny: Compositional Methods

In this subsection, we present some experimental results concerning the alignment-free compositional methods presented in Section 16.4. These methods, based on the notion of subword composition of sequences, have become significant for biological data mining thanks to extensive experimentation. In particular, they have been tested extensively on protein sequences [16, 69, 86].

Wu *et al.* [86] apply their method to infer the phylogeny footprint of 64 vertebrates, with 13 homologous proteins for each species, and 99 microbes. In particular, they show that the constructed phylogeny on the first dataset is largely consistent with the taxonomy tree. In fact, all perissodactyls, carnivores, cetartiodactyls, rodents, primates, noneutherians, birds, reptiles, bony fish, and cartilaginous fish are grouped together correctly. Concerning the second dataset, each of the 99 microbes is represented by its complete set of protein sequences. By comparing the CCV-based phylogeny and the taxonomy tree, they show that they are similar in most branches.

Qi *et al.* [69] use their distance to build phylogenetic trees of 109 organisms, including 16 Archaea, 87 Bacteria, and 6 Eukaryota. Qi *et al.* study how the topology of the phylogenetic tree they build varies with the parameter $k$. Remarkably, their experiments show that the topology of the tree exhibits less and less variation as $k$ increases. Moreover, for the taxa they examine, such a topology becomes stable for $k = 5$ and 6. In general, their phylogenetic trees support the small subunit (SSU) rRNA "tree of life" in its overall structure and in many details. Moreover, even if their trees and the SSU rRNA tree are based on nonoverlapping parts of the genomic data, namely, the RNA segments and the protein-coding sequences, and they are obtained by using entirely different ways of inferring distances between species, then they lead to basically consistent results.

Apostolico and Denas [16] report experiments on a dataset that consists of two Eukaryota, four Archea, of which two Euryarchaeota and two Crenarchaeota, four

Bacteria, of which three Proteobacteria and one Thermotogae. The distance computations based on all $k$-words produce unreliable trees as soon as $k > 7$. At low level taxa (*i.e.*, lower levels of classification discrimination), trees based on fixed-length $k$-words and maximal $k$-words are consistent, as they both correctly group together Eukaryota, Proteobacteria, Euryarcheota, and Crenarchaeota. However, at higher level taxa, the distance based on maximal $k$-words seems to be more stable. In fact, it groups Euryarcheota and Crenarchaeota in all cases, whereas with fixed-length $k$-mers, this holds only for $k \leq 9$. All methods fail grouping Thermotogae with Proteobacteria, a deficiency that might be attributable to the absence of other organisms from the dataset.

Finally, Apostolico and Denas consider a sample dataset comprising seven Firmicutes, one Fuso, one Thermotogae and one Aquificae. Even if their experiments are on this limited dataset, it seems that their distances based on fixed-length $k$-words perform well for moderate values of $k$, whereas it seems to loose stability with "distant" organisms and resolution with "close" ones for larger values of $k$.

### 16.6.3 CIS Regulatory Modules

Regulatory sequence comparison can prove vital for the *ab initio* discovery of cis-regulatory modules (CRMs) with a common function. A CRM may be defined as a contiguous noncoding sequence that contains multiple transcription factor binding sites and drives many aspects of gene expression profiles. If a set of coregulated genes in a single species is given, then it is important to find in their upstream and downstream regions (called "control regions") the CRMs that mediate the common aspect of their expression profiles. The control regions may be tens of Kbp long for each gene (especially for metazoan genomes), whereas the CRMs to be discovered are often only hundreds of bp long. One therefore must search in the control regions for subsequences (the candidate CRMs) that share some functional similarity. The CRM search algorithm thus requires a method that can discern functional similarity among candidate CRMs based on their sequence similarity. Also, because the different CRMs are only functionally related, and not orthologous, it is useful that the comparison method is alignment-free [47].

Papers in this chapter that make evaluations on functionally related regulatory sequences are essentially [26,47]. Both papers contain experiments on the following seven datasets, well studied by Gallo *et al.* [36] and accurately described in [47]:

- FLY BLASTODERM: 82 CRMs with expression in the blastoderm-stage embryo of the fruitfly, *Drosophila melanogaster*
- FLY PNS: 23 CRMs (average length 998 bp) driving expression in the peripheral nervous system of the fruitfly
- FLY TRACHEAL: 9 CRMs (average length 1220 bp) involved in the regulation of the tracheal system of the fruitfly
- FLYEYE: 17 CRMs (average length 894 bp) expressing in the *Drosophila* eye
- HUMAN MUSCLE: 28 human CRMs (average length 450) regulating muscle-specific gene expression

- HUMAN LIVER: 9 CRMs (average length 201) driving expression specific to the human liver
- HUMAN HBB: 17 CRMs (average length 453) regulating the HBB complex

Kantorovitz *et al.* perform extensive tests with the $D_2$ z-score on the listed tissue-specific families of known enhancers. Their results show that the $D_2$ z-score accurately can discriminate functionally related CRMs from unrelated sequence pairs, and orthologous CRMs from orthologous nonregulatory noncoding sequences. Moreover, they also compared the $D_2$z with five other scores (euclidean distance, empirical relative entropy, van Helden's Poisson scores [77], Pearson's correlation coefficient, and cosine function). All scores are run with word lengths $k = 5, 6$. The $D_2$ z-score and the Poisson score are run with background models of Markov order $0, 1, 2$. The authors report that the $D_2$ z-score outperforms all other scores in each of the datasets analyzed, with the Poisson score being the next best method in five datasets.

Analogous experiments are presented in [26], in which Dai *et al.* make use of the statistical measures they introduce to evaluate whether functionally or evolutionary related sequence pairs are scored better than unrelated pairs of sequences randomly chosen from the genome. They analyze the seven datasets described and compare similarity measures that satisfy the symmetry condition. The measures they consider are: $S_1.k.r$, $S_2.k.r$, euclidean distance, empirical relative entropy, $D_2$, $D_2z$, SimMM, Pearson's correlation coefficient, cosine function, and similarity measures based on alignment, which are NeedlemanWunsch for global alignment or SmithWaterman for local alignment. All statistical measures based on $k$-word distributions run with $k$-word from two to eight, and the similarity measures based on Markov model run with Markov order $r$ from zero to two. The experiments on these seven datasets show that $S_2.k.r$ performs significantly better than other measures in six experiments because it incorporates the $k$-word information into the Markov model directly. It is important to point out that, because the different CRMs are only functionally related and not orthologous, the CRM search algorithm requires a method that can discern functional similarity among candidate CRMs based on their sequence similarity. ROC analysis on the seven datasets shows that the alignment-free methods give better results than alignement-based ones in the evaluation of functionally related regulatory sequences.

### 16.6.4 DNA Sequence Dependencies

This subsection is devoted to describe some experimental results presented by Aktulga *et al.* in [12]. The focus of their work is on the development of reliable and precise information-theoretic methods that determine whether two biological sequences are statistically dependent or not.

They make two different kinds of experiments based on the application of the empirical mutual information. More precisely, they first show that this notion can be used effectively to identify statistical dependence between regions of the maize zmSRp32 gene [37], which belongs to a group of genes functionally homologous to the human alternative splicing factor/splicing factor 2 (ASF/SF2). This gene may be involved in alternative processing (splicing) of pre-mRNA transcripts. Second, the authors show how this methodology can be applied to the problem of identifying short tandem repeats (STRs) in genomic sequences, which represents an important task in genetic profiling.

Concerning the first experiment, Aktulga *et al.* present their empirical findings for the problem of detecting statistical dependency between different parts in a DNA sequence by making extensive numerical experiments carried out on certain regions of the maize zmSRp32 gene. To understand and quantify the amount of correlation between different parts of this gene, they compute the mutual information between all functional elements including exons, introns, and the 5′ untranslated region (5′UTR). Their findings show the existence of a biological connection between the 5′UTR in zmSRp32 and its alternatively spliced exons. The UTRs are multifunctional genetic elements that control gene expression by determining mRNA stability and efficiency of mRNA translation. Like in the zmSRp32 maize gene, they can provide multiple alternatively spliced variants for more complex regulation of mRNA translation. Therefore, they observe that the maize zmSRp32 5′UTR contains information that could be used in the process of alternative splicing. This is stressed by the fact that the value of the empirical mutual information between 5′UTR and the DNA sequences that encode alternatively spliced elements is significantly greater than zero.

Concerning the second experiment, Aktulga *et al.* examine the performance of empirical mutual information statistic on the problem of detecting STRs in genomic sequences. STRs, usually found in noncoding regions, are made of back-to-back repetitions of a sequence that is at least two bases long and is generally shorter than 15 bases. Their short lengths let STRs survive mutations well.

Many algorithms exist for the problem of detecting STRs in long DNA strings with no prior knowledge about the size and the pattern of repetition [41]. These algorithms mostly are based on pattern matching, and they all have high time complexity. Moreover, when the query string is a DNA segment that contains many errors resulting from mutations, the problem becomes even harder. To overcome these limitations, the authors propose a statistical approach using an adaptation of their method. More precisely, results in the paper prove that their methodology is very effective at detecting the presence of STRs, although at first glance, it may seem like it cannot provide precise information about their start-end positions and their repeat sequences. Their method can be seen as an initial filtering step that has to be combined with an exact pattern matching algorithm.

## 16.7 DATASETS AND SOFTWARE FOR EXPERIMENTAL ALGORITHMICS

In this section, we present datasets that we believe to be of interest for comparative analysis of similarity measures, as they are used for classification and phylogenetic studies. Indeed, several benchmark datasets of nonhomologous protein structures and genomes have been assembled in the last few years (*e.g.*, [21, 71, 75, 84]), by researchers in this area, but there is not a common line in the use of them. Therefore, we limit our presentation to a kernel of datasets that seem to be used the most for benchmarking purposes. Moreover, although most papers in this area report experimental results, there are very few publicly available software libraries that one can use for benchmarking and comparative studies. We provide the ones that are prominent in this area.

### 16.7.1 Datasets

Among several datasets of nonhomologous protein structures we have identified the *Chew-Kedem* dataset, the guanine nucleotide-binding proteins (*G-proteins*), subsets of the Clusters of Orthologus Groups (*COG*) *database* and the *GH2 family*. We also have included in this presentation a benchmark dataset of 15 complete unaligned mitochondrial genomes, referred to as the *Apostolico dataset*. Some details about them are provided next.

Note that for datasets of protein structures, it is possible to consider several alternative representations for the structure. Besides the standard representation of amino acid sequences in FASTA format [63], it is also possible to use a text file consisting of the ATOM lines in the Protein Data Bank (PDB) entry for the protein domain, the topological description of the protein domain as a TOPS string of secondary structure elements [34, 39, 82, 83], and the complete TOPS string with the contact map. The TOPS model is based on secondary structure elements derived using DSSP [44] plus the chirality algorithm of [83].

The *Chew-Kedem dataset* [21] is a *de facto* standard in this area as it has been used as a benchmark in many studies related to this chapter (*e.g.*, [51]). It consists of 36 protein domains drawn from PDB entries of three classes (alpha-beta, mainly-alpha, mainly-beta), which are listed as follows.

- alpha-beta: 1aa900, 1chrA1, 1ct9A1, 1gnp00, 1qraA0, 2mnr01, 4enl01, 5p2100, 6q21A0, and 6xia00
- mainly beta: 1cd800, 1cdb00, 1ci5A0, 1hnf01, 1neu00, 1qa9A0, and 1qfoA0
- mainly alpha: 1ash00, 1babA0, 1babB0, 1cnpA0, 1eca00, 1flp00, 1hlb00, 1hlm00, 1ithA0, 1jhgA0, 1lh200, 1mba00, 1myt00, 2hbg00, 2lhb00, 2vhb00, 2vhbA0, 3sdhA0, and 5mbn00

The files of this dataset are provided at [6]. For the classification of protein domains, CATH classification of proteins [62] is assumed to provide the gold standard [2].

Good benchmark datasets to assess the accuracy of similarity measures and classification algorithms in grouping protein sequences according to their functional annotation and biological classification are based on the phylogenetic classification of proteins encoded in complete genomes. That is commonly referred to as the COGs database [8]. The COGs were identified by comparing protein sequences encoded in complete genomes, representing major phylogenetic lineages. Each COG consists of individual proteins or groups of paralogs from at least three lineages and thus corresponds to an ancient conserved domain. Six randomly generated subsets of the COG database were made available by Kelil *et al.* [48]. The FASTA files of these subsets are provided at [8, 48]. The classification of the COG database [8] is taken to be the gold standard.

Another family of proteins that has been used in a considerable number of publications and is a good reference classification is represented by the G-proteins family. It is well known that G-proteins is a family of proteins "easy to align." From a biological point of view, G-proteins are a family of proteins involved in second-messenger cascades, and they belong to the larger group of enzymes called guanosine triphosphate (GTPases). G-proteins are so called because they function as "molecular switches," alternating between an inactive guanosine diphosphate (GDP) and active GTP bound state, ultimately going on to regulate downstream cell processes.

A dataset containing 381 protein sequences selected from the G-proteins and receptor activity-modifying proteins (RAMPs) family was made available by Kelil *et al.* [48]. The FASTA files of these subsets are provided at [4, 48]. RAMPs are a class of proteins that interact with and modulate the activities of several Class B G-protein-coupled receptors (GPCRs) including the receptors for secretin, calcitonin (CT), glucagon, and vasoactive intestinal peptide (VIP). There are three distinct types of RAMPs, denoted RAMP1, RAMP2, and RAMP3, each encoded by a separate gene. The classification is according to the GPCRIPDB Data Base (a molecular-specific information system for GPCR interacting partners (G-proteins and RAMPs) [5], and it is taken as the gold standard.

The glycoside hydrolase family 2 (GH2) is a multidomain protein family whose members are known to be "hard to align." In fact, no definitive multiple alignment of this family is available yet. For this reason, it is a particularly challenging family of sequences for multiple alignment algorithms. From a biological point of view, the glycoside hydrolases are a widespread group of enzymes that hydrolyse the glycosidic bond between two or more carbohydrates or between a carbohydrate and a noncarbohydrate moiety. Among glycoside hydrolases families, the GH2 family includes enzymes that perform five distinct hydrolytic reactions. With respect to known biochemical activities, we can distinguish among the following:

- beta-galactosidases
- beta-mannosidases
- beta-glucuronidases
- exo-beta-glucosaminidase
- mannosylglycoprotein endo-beta-mannosidase (in plants)

A dataset of 316 protein sequences belonging to the GH2 family selected from the Carbohydrate-Active Enzymes (CAZy) database was made available by. Kelil *et al.* [48]. The FASTA files of these subsets are provided at [9, 48]. The CAZy

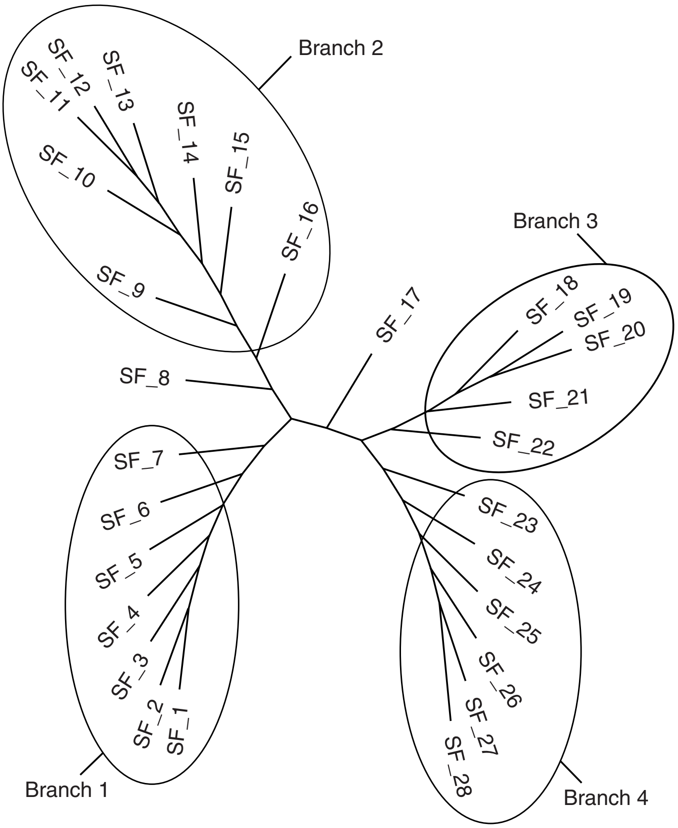


**Figure 16.2** The gold tree of the GH2 family.

database describes the families of structurally related catalytic and carbohydrate-binding modules (or functional domains) of enzymes that degrade, modify, or create glycosidic bonds. The phylogenetic tree shown in Figure 16.2 and given in [48, Figure 6] is taken to be the gold standard according to the following division:

- Branch 1 corresponds to "$\beta$-galactosidases" activities
- Branch 3 corresponds to "$\beta$-mannosidase" activities
- Branch 4 corresponds to "$\beta$-glucuronidase" activities
- Branch 2 contains enzymes labeled as "putative $\beta$-galactosidases" in databases
- the subfamily SF − 8 includes 22 "exoglucosaminidase" and "endo-mannosidase" activities
- the subfamily SF − 17 includes 19 sequences labeled "$\beta$-galactosidases" in databases. Although the branch 1 "$\beta$-galactosidases" consist of five modules, known as the "sugar binding domain," the "immunoglobulin-like $\beta$-sandwich," the "$(\alpha\beta)$8-barrel," the "$\beta$-gal small-N domain," and the "$\beta$-gal small-C

domain," the members of subfamily 17 lack the last two of these domains, which makes them more similar to "$\beta$-mannosidases" and "$\beta$-glucuronidases." These enzymes are distinct from those of branch 1, and their separate localization is justified,

- the Subfamily SF − 28 contains a sequence that is a putative glycosyltransferase, and hence, it is not a member of GH2 family. In the phylogenetic tree in Figure 16.2 this sequence is in the SF − 28 that is contained in branch 4.

The *Apostolico dataset* [15] consists of complete unaligned mitochondrial genomes of the following 15 mammals from GenBank: human (*Homo sapiens* [GenBank: V00662]), chimpanzee (*Pan troglodytes* [GenBank:D38116]), pigmy chimpanzee (*Pan paniscus* [GenBank:D38113]), gorilla (*Gorilla gorilla* [GenBank:D38114]), orangutan (*Pongo pygmaeus* [GenBank:D38115]), gibbon (*Hylobates lar* [GenBank: X99256]), sumatran orangutan (*Pongo pygmaeus abelii* [GenBank:X97707]), horse (*Equus caballus* [GenBank:X79547]), white rhino (*Ceratotherium simum* [GenBank:Y07726]), harbor seal (*Phoca vitulina* [GenBank:X63726]), gray seal (*Halichoerus grypus* [GenBank:X72004]), cat (*Felis catus* [Gen- Bank:U20753]), finback whale (*Balenoptera physalus* [GenBank:X61145]), blue whale (*Balenoptera musculus* [GenBank:X72204]), rat (*Rattus norvegicus* [Gen- Bank:X14848]), and house mouse (*Mus musculus* [Gen- Bank:V00711]). The dataset, in FASTA format, is provided in [6]. It is a standard for the assessment of performance of similarity measures and phylogenetic algorithms. Although there is no gold standard for the entire tree, biologists suggest the following grouping for this case:

- *Eutheria-Rodens*: house mouse and rat
- *Primates*: chimpanzee, gibbon, gorilla, human, orangutan, pigmy chimpanzee, and sumatran orangutan
- *Ferungulates*: blue whale, finback whale, gray seal, harbor seal, horse, and white rhino

However, one can assume the NCBI Taxonomy [7] as the gold standard. For convenience of the reader, that tree is reported in Figure 16.3.

### 16.7.2 Software

As supplementary material to their fundamental paper on alignment-free methods [78], Vinga and Almeida provided a Matlab library that implements some basic methods mostly based on the evaluation of distances between vectors. For a string $A$, each entry in its characteristic vector corresponds to a k-word $w$, and the value of that entry corresponds to the number of occurrences of $w$ in $A$. The software is available at [1]. A comparative study, as well as novel methods based on characteristic vectors, is presented in [79].

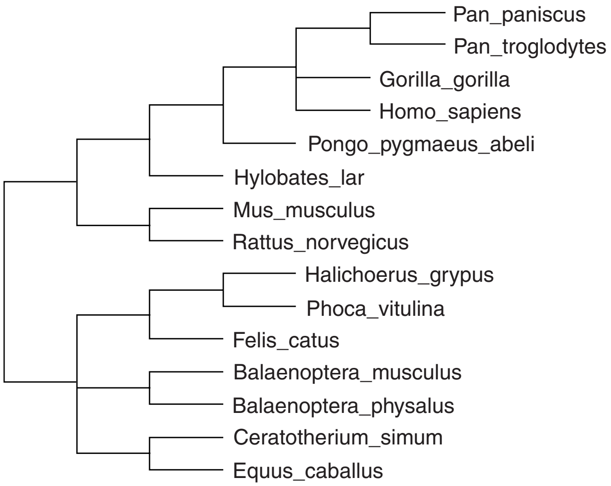


**Figure 16.3** The NCBI philogenetic tree of the Apostolico dataset.

As a supplementary contribution to their extensive study of the Universal Similarity Metric (USM) as it applies to biological data mining [30], Ferragina *et al.* provide an entire library of more than 20 data compression methods together with software to compute the various incarnations of the USM via those compressors. The entire system is available at [6]. Based on the USM, Barthel *et al.* [17] provide an entire decision support system for the classification of proteins. The corresponding web server, nicknamed Pro(CKSI), can be found at [11].

Apostolico and Denas provide both public software and a web service to evaluate distances between strings that are based on composition vectors [3]. Finally, Kantorovitz *et al.* also provide a web server as well as a publicly available software library for their $D_2$ measure [10].

## 16.8 CONCLUSIONS

This chapter surveys some alignment-free distances between biological sequences, chosen among the several ones in the specialistic literature, that are perceived as representative of the novel techniques that have been devised in the past few years. Given the availability of complete genomes of various organisms, sequence alignment is not suitable to measure events and mutations that involve longer segments of genomic sequences because it considers only local mutations of the genome. Furthermore, a major challenge is to define alignment-free distances that make use of algorithms and structures able to handle the growing mass of biological data.

We have described some alignment-free methods, ranging from combinatorics to information theory that have been applied to genomic sequences or that seem to be scalable at the genomic level. In particular, we considered distances making explicit use of information measures, combinatorial-based measures, methods based on the notion of subword composition, and finally, exact word matches methods. Moreover, each approach has been validated experimentally on a collection of relevant biological datasets, such as proteomic and genomic sequences, mitochondrial genomes, and proteins. From those experimental results, it is possible to deduce that some methods are efficiently scalable when the datasets become large and the sequences become long. For the others, the genomic-wide experimentation seems to be within reach.

Finally, we presented some datasets for comparative analysis of similarity measures. Actually, several benchmark datasets of nonhomologous protein structures and genomes have been collected in the last few years, but they are not largely used. So, we provided a kernel of datasets that seem to be the most used for benchmarking purposes. We also provided the most prominent publicly available software libraries in this area that can be used for benchmarking and comparative studies.

## REFERENCES

1. Alignment-Free Vinga and Almeida Library. http://bioinformatics.musc.edu/resources.html.
2. CATH DataBase. http://www.cathdb.info/.
3. Galaxy/Laboratory for Bioinformatics-Padova. http://bcb.dei.unipd.it/.
4. GPCRIPDB: Information system for GPCR interacting proteins. http://www.gpcr.org.
5. GPCRIPDB: Information system for GPCR interacting proteins. http://www.gpcr.org/GPCRIP/multali/multali.html.
6. Kolmogorov Library Supplementary Material Web Page. http://www.math.unipa.it/ raffaele/kolmogorov/datasets/.
7. NCBI Taxonomy. http://www.ncbi.nlm.nih.gov/Taxonomy/.
8. Phylogenetic classification of proteins encoded in complete genomes. http://www.ncbi.nlm.nih.gov/COG/.
9. The carbohydrate-active enzymes (CAZy) database. http://www.cazy.org/.
10. The D2Z Web Site. http://veda.cs.uiuc.edu/cgi-bin/d2z/download.pl.
11. The ProCKSI-Server. http://www.procksi.net/.
12. H. Metin Aktulga, I. Kontoyiannis, L.A. Lyznik, L. Szpankowski, A.Y. Grama, and W. Szpankowski. Identifying statistical dependence in genomic sequences via mutual information estimates. *EURASIP J Bioinformatics Syst Biol*, 2007.
13. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215:403–410, 1990.
14. S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped blast and psiblast: A new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–3402, 1997.
15. A. Apostolico, M. Comin, and L. Parida. Mining, compressing and classifying with extensible motifs. *Algorithm Mol Biol*, 1:4, 2006.
16. A. Apostolico and O. Denas. Fast algorithms for computing sequence distances by exhaustive substring composition. *Algorithms Mol Biol*, 3(1):13, 2008.

17. D. Barthel, J.D. Hirst, J. Blażewicz, E.K. Burke, and N. Kransnogor. ProCKSI: A decision support system for protein (structure) comparison, knowledge, similarity and information. *BMC Bioinformatics*, 8:416, 2007.

18. B.E. Blaisdell. A measure of the similarity of sets of sequences not requiring sequence alignment. *Proc Natl Acad Sci*, 83:5155–5159, 1986.

19. M. Burrows and D.J. Wheeler. A block sorting data compression algorithm. Technical report, DIGITAL System Research Center, 1994.

20. Y. Cao, A. Janke, P.J. Waddell, M. Westerman, O. Takenaka, S. Murata, N. Okada, S. Pääbo, and M. Hasegawa. Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. *J Mol Evol*, 47:307–322, 1998.

21. L.P. Chew and K. Kedem. Finding the consensus shape for a protein family. *Algorithmica*, 38:115–129, 2003.

22. R. Cilibrasi and P.M.B. Vitányi. Clustering by compression. *IEEE Trans Inform Theory*, 51(4):1523–1545, 2005.

23. J.R. Cole, B. Chai, T.L. Marsh, R.J. Farris, Q. Wang, S.A. Kulam, S. Chandra, D.M. McGarrell, T.M. Schmidt, G.M. Garrity, and J.M. Tiedje. The ribosomal database project (rdp-ii: Previewing a new autoaligner that allows regular updates and the new prokaryotic taxonomy. *Nucleic Acids Res*, 31:442–443, 2003.

24. E. Corel, R. El Fegalhi, F. Gérardin, M. Hoebeke, M. Nadal, A. Grossmann, and C. Devauchelle. Local similarities and clustering of biologicalsequences: New insights from n-localdecoding. *The First International Symposium on Optimization and Systems Biology (OSB07)*, Beijing, China, 2007, pp. 189–195.

25. T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.

26. Q. Dai, Y. Yang, and T. Wang. Markov model plus-word distributions: A synergy that produces novel statistical measures for sequence comparison. *Bioinformatics*, 24(20):2296–2302, 2008.

27. M.O. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5 of *National Biomedical Research Foundation*, Washington, DC, 1978, pp. 345–352.

28. G. Didier, L. Debomy, M. Pupin, M. Zhang, A. Grossmann, C. Devauchelle, and I. Laprevotte. Comparing sequences without using alignments: Application to hiv/siv subtyping. *BMC Bioinformatics*, 2(8:1), 2007.

29. G. Didier, I. Laprevotte, M. Pupin, and A. Hénaut. Local decoding of sequences and alignment-free comparison. *J Comput Biol*, 13(8):1465–1476, 2006.

30. P. Ferragina, R. Giancarlo, V. Greco, G. Manzini, and G. Valiente. Compression-based classification of biological sequences and structures via the universal similarity metric: experimental assessment. *BMC Bioinformatics*, 8, 2007.

31. P. Ferragina, R. González, G. Navarro, and R. Venturini. Compressed text indexes: From theory to practice! *ACM J Exp Algorithmics*, 2008.

32. G. Fichant and C. Gautier. Statistical method for predicting protein coding regions in nucleic acid sequences. *Comput Appl Biosci*, 3:287–295, 1987.

33. N.J. Fine and H.S. Wilf. Uniqueness theorem for periodic functions. *Proc Am Math Soc*, 16:109–114, 1965.

34. T.P. Flores, D.M. Moss, and J.M. Thornton. An algorithm for automatically generating protein topology cartoons. *Protein Eng Des Sel*, 7:31–37, 1994.

35. S. Foret, M. Kantorovitz, and C. Burden. Asymptotic behaviour and optimal word size for exact and approximate word matches between random sequences. *BMC Bioinformatics*, 7(Suppl 5):S21, 2006.

36. S.M. Gallo, L. Li, Z. Hu, and M.S. Halfon. *EDfly:* a regulatory element database for *rosophila*. *Bioinformatics*, 22(3):381–383, 2006.

37. H. Gao, W.J. Gordon-Kamm, and L.A. Lyznik. Asf/sf2-like maize pre-mrna splicing factors affect splice site utilization and their transcripts are alternatively spliced. *Gene*, 339(1-2):25–37, 2004.

38. R. Giancarlo, D. Scaturro, and F. Utro. Textual data compression in computational biology: a synopsis. *Bioinformatics*, 25(13):1575–1586, 2009.

39. D.R. Gilbert, D.R. Westhead, N. Nagano, and J.M. Thornton. Motif-based searching in tops protein topology databases. *Bioinformatics*, 15:317–326, 1999.

40. O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162:705–708, 1982.

41. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, U.K., 1997.

42. S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89:10915–10919, 1992.

43. A.C.-C. Yang S.-S. Hseu, H.-W. Yien, A. L. Goldberger, and C.-K. Peng. Linguistic analysis of the human heartbeat using frequency and rank order statistics. *Phys Rev Lett*, 90(10):108103, 2003.

44. W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

45. M.R. Kantorovitz, H.S. Booth, C.J. Burden, and S.R. Wilson. Asymptotic behavior of k-word matches between two random sequences. *J Appl Probab*, 44:788–805, 2007.

46. M.R. Kantorovitz, C.J. Burden, and S.R. Wilson. Approximate word matches between two random sequences. *Ann Appl Probab*, 18:1–21, 2008.

47. M.R. Kantorovitz, G.E. Robinson, and S. Sinha. A statistical method for alignment-free comparison of regulatory sequences. *ISMB/ECCB (Supplement of Bioinformatics)*, 2007, pp. 249–255.

48. A. Kelil, S. Wang, R. Brzezinski, and A. Fleury. Cluss: Clustering of protein sequences based on a new similarity measure. *BMC Bioinformatics*, 8:286, 2007.

49. M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J Mol Evol*, 16:111–120, 1980.

50. J. Kleffe and M. Borodovsky. First and second moment of counts of words in random texts generated by markov chains. *Comput Appl Biosci*, 8(5):433–441, 1992.

51. N. Krasnogor and D.A. Pelta. Measuring the similarity of protein structures by means of the universal similarity metric. *Bioinformatics*, 20(7).

52. M. Li, J. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154, 2001.

53. M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi. The similarity metric. *IEEE Trans Inform Theory*, 50(12):3250–3264, 2004.

54. W. Li, W. Fang, L. Ling, J. Wang, Z. Xuan, and R. Chen. Phylogeny based on whole genome as inferred from complete information set analysis. *J Biol Phys*, 28(3):439–447, 2002.

55. R.A. Lippert, H. Huang, and M.S. Waterman. Distributional regimes for the number of k-word matches between two random sequences. *Proc Natl Acad Sci U S A*, 99(22):13980–13989, 2002.

56. G. Lu, S. Zhang, and X. Fang. An improved string composition method for sequence comparison. *BMC Bioinformatics*, 9(Suppl 6):S15, 2008.

57. S. Mantaci, A. Restivo, G. Rosone, and M. Sciortino. A new combinatorial approach to sequence comparison. *Theor Comput Syst*, 42(3):411–429, 2008.

58. F. Mignosi, A. Restivo, and M. Sciortino. Words and forbidden factors. *Theor Comput Sci*, 273(1-2):99–117, 2002.

59. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comput Surv*, 39:2, 2007.

60. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48:443–453, 1970.

61. H.H. Otu and K. Sayood. A new sequence distance measure for phylogenetic tree construction. *Bioinformatics*, 19(16):2122–2130, 2003.

62. F. Pearl, A. Todd, I. Sillitoe, M. Dibley, O. Redfern, T. Lewis, C. Bennett, R. Marsden, A. Grant, D. Lee, et al. The cath domain structure database and related resources gene3d and dhs provide comprehensive domain family information for genome analysis. *Nucleic Acids Res*, 33(D):D247–D251, 2005.

63. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci*, 85(8):2444–2448, 1998.

64. W.R. Pearson. Rapid and sensitive sequence comparison with fastp and fasta. *Meth Enzymol*, 183:63–98, 1990.

65. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85:2444–2448, 1988.

66. P. Petrilli. Classification of protein sequences by their dipeptide composition. *Comput Appl Biosci*, 9(2):205–209, 1993.

67. T.D. Pham and J. Zuegg. A probabilistic measure for alignment-free sequence comparison. *Bioinformatics*, 20:3455–3461, 2004.

68. J. Qi, H. Luo, and B. Hao. Cvtree: A phylogenetic tree reconstruction tool based on whole genomes. *Nucleic Acids Res*, 32:45–47, 2004.

69. J. Qi, B. Wang, and B.-I. Hao. Whole proteome prokaryote phylogeny without sequence alignment: A k-string composition approach. *J Mol Evol*, 58:111, 2004.

70. D. Sankoff and J.B. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

71. M. Sierk and W. Person. Sensitivity and selectivity in protein structure comparison. *Protein Sci*, 13(3):773–785, 2004.

72. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147:195–197, 1981.

73. G.W. Stuart, K. Moffett, and S. Baker. Integrated gene and species phylogenies from unaligned whole genome protein sequences. *Bioinformatics*, 18:100–108, 2002.

74. G.W. Stuart, K. Moffett, and J.J. Leader. A comprehensive vertebrate phylogeny using vector representations of protein sequences from whole genomes. *Mol Biol Evol*, 19:554–562, 2002.

75. B. Thiruv, G. Quon, S.A. Saldanha, and B. Steipe. Nh3d: A reference dataset of non-homologous protein structures. *BMC Struct Biol*, 5:12, 2005.

76. I. Ulitsky, D. Burstein, T. Tuller, and B. Chor. The average common substring approach to phylogenomic reconstruction. *J Comput Biol*, 13(2):336–350, 2006.

77. J. van Helden. Metrics for comparing regulatory sequences on the basis of pattern counts. *Bioinformatics*, 20(3):399–406, 2004.

78. S. Vinga and J.S. Almeida. Alignment-free sequence comparison—a review. *Bioinformatics*, 19(4):513–523, 2003.

79. S. Vinga, R. Gouveia-Oliveira, and J.S. Almeida. Comparative evaluation of word composition distances for the recognition of scop relationships. *Bioinformatics*, 20(2):206–215, 2004.

80. J. Wang and X. Zheng. WSE, a new sequence distance measure based on word frequencies. *Math Biosci*, 215:78–83, 2008.

81. M.S. Waterman. *Introduction to Computational Biology. Maps, Sequences and Genomes*. Chapman Hall, London, 1995.

82. D.R. Westhead, D.C. Hutton, and J.M. Thornton. An atlas of protein topology cartoons available on the world wide web. *Trends Biochem Sci*, 23:35–36, 1998.

83. D.R. Westhead, T. Slidel, T. Flores, and J.M. Thornton. Protein structural topology: Automated analysis and diagrammatic representations. *Protein Sci*, 8(4):897–904, 1999.

84. J.M. Word, S.C. Lovell, T.H. LaBean, H.C. Taylor, M.E. Zalis, B.K. Presley, J.S. Richardson, and D.C. Richardson. Visualizing and quantifying molecular goodness-of-fit: Small-probe contact dots with explicit hydrogen atoms. *J Mol Biol*, 285(4):1711–1733, 1999.

85. T.-J. Wu, Y.-C. Hsieh, and L.-A. Li. Statistical measures of dna sequence dissimilarity under markov chain models of base composition. *Biometrics*, 57(2):441–448, 2001.

86. X. Wu, X.-F. Wan, G. Wu, D. Xu, and G. Lin. Phylogenetic analysis using complete signature information of whole genomes and clustered neighbour-joining method. *IJBRA*, 2(3):219–248, 2006.

# 17

# *IN SILICO* METHODS FOR THE ANALYSIS OF METABOLITES AND DRUG MOLECULES

Varun Khanna and Shoba Ranganathan

## 17.1 INTRODUCTION

### 17.1.1 Chemoinformatics and "Drug-Likeness"

During the past decades, paralleling the exponential data flow from the -omic sciences, there has been a steep rise in information technology approaches to organize and mine the enormous data generated. Chemoinformatics [1, 2] belongs to a large family of informatic sciences that include contemporary areas like bioinformatics [3], immunoinformatics [4], biodiversity informatics [5], and biomedical informatics [6]. *Chemoinformatics* is the application of information technology to address problems in the realm of chemistry and to tackle the massive chemical space, comprising natural products, synthetic organic compounds, drugs, and toxins. At its inception, chemoinformatics was restricted to the *in silico* drug discovery process [7]. Subsequently, the definition has been broadened to incorporate the storage, management, and retrieval of chemical data, reaction and spectral analysis, visualization, similarity and diversity analyses, virtual library design, molecular modeling, and structure activity/property relationships studies. Gasteiger [8], Leach and Gillet [9], and more recently, Agrafiotis *et al.* [10] provide comprehensive accounts on chemoinformatics. The exponential growth of biological and chemical data has raised crucial

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya
Copyright © 2011 John Wiley & Sons, Inc.

challenges for maintaining and searching the databases for knowledge mining. Structure representation, data handling, and searching of chemical databases are key aspects of chemoinformatics, with searching of chemical databases being particularly significant in drug discovery programs [11].

The drug discovery and optimization process has undergone a rapid change in the last two decades. With the discovery of two complementary technologies in the early 1990s, viz. high-throughput screening and combinatorial synthesis, pharmaceutical companies now can synthesize and assay a vast number of compounds per target per year [12]. However, an exhaustive search of chemical space is not feasible because of the enormity of the task, as a conservative estimate of known small molecules is of the order of $10^{60}$ [13]. As a result, an early *in silico* prediction of pharmacological properties of potential drug candidates is becoming increasingly popular [14] to cut down the time required to bring a drug into the market. Low-cost computational technologies, such as similarity searching by constructing pharmacophore models, diversity oriented synthesis, and molecular modeling are finding widespread applications in the drug discovery process. There also has been an increased effort to produce focused or directed libraries that are biased toward a specific target, structural class, or known pharmacophore [15] over universal libraries, which are target-independent.

Subsequently, the concept of "drug-like" (DL) [16, 17] or "lead-like" [18] molecules evolved and various models were developed to recognize DL compounds from a diverse set of molecules [19]. Walters *et al.* [17] summarized *"drug-likeness"* as the tendency of a molecule to contain functional groups and/or physical properties consistent with the majority of known drugs. The credit for popularizing the concept of DL substances goes to Lipinski, Murcko, and researchers at Pfizer and Vertex [20], who performed a statistical analysis of 2200 drugs from the World Drug Index (WDI). They established certain threshold values that seem to be valid for most drugs known at that time. In 2008, Ertl *et al.* [21] introduced a similar measure called Natural Product (NP)-likeness score to characterize a molecule and thereby distinguish it from drugs and synthetic compounds. Similarly, with the growing knowledge of biochemical pathways and the cognate metabolites, several researchers have recommended metabolite-likeness [22, 23] as one of the main criteria for drug design. Dobson *et al.* [22] compared different molecular properties among human metabolites, drugs, and "predrugs" (precursor drug molecules) and concluded that although metabolites are a distinct class of compounds, they share a significant percentage of property space with drugs. They further suggested that metabolite-likeness may be used as a filter for designing drugs that are functionally similar to metabolites, leading to better absorption, distribution, metabolism, elimination, toxicology (ADMET) properties, which are critical for the delivery and uptake of drugs *in vivo*.

All drug discovery methods aim to identify properties or group of features, known as descriptors that are necessary for designing drugs with reduced toxicity, improved potency, selectivity, and bioavailability. Commonly used descriptors, data sources, and methods available for chemoinformatics research with particular emphasis on drug discovery are discussed in the following sections. Starting with the various forms to represent chemical structures, we briefly survey the freely available and most commonly used small molecule databases and then introduce the algorithms available for drug research. Current trends, future directions, and some key suggestions that could solve existing problems are presented.

## 17.2 MOLECULAR DESCRIPTORS

The set of features or characteristics that describes a molecule to the best approximation are called *molecular descriptors*. Descriptors can be calculated using chemical structure (conformations, configurations, and constitution) or the properties (physical, chemical, or biological) of the small molecules [24]. Classically, descriptors are categorized into three groups based on complexity and encoded information.

### 17.2.1 One-Dimensional (1-D) Descriptors

One-dimensional (1-D) representations of a molecule generate the simplest known descriptors that include physicochemical properties (log P and molecular weight), count of chemical features (number of hydrogen bond donors and number of rotatable bonds), molecular codes, and so on.

***17.2.1.1 Physicochemical Properties.*** The simplest types of descriptors are calculated physicochemical properties of molecules (log P, atom counts, and molecular weight). These often are referred to as *whole molecule descriptors* as they describe the properties of the complete molecule. The advantage of these descriptors is that they are easily calculable and can be predicted based on molecular structure alone [25].

***17.2.1.2 Molecular Codes.*** *Molecular codes* are the descriptors calculated from linear notations such as molecular formula, 1-D line notations such as *Simplified Molecular Input Line Entry Specification* (SMILES) representation [26], the *International Union of Pure and Applied Chemistry* (IUPAC) name, and more recently, *International Chemical Identifier* codes (InChI) developed in collaboration between IUPAC and NIST (*National* Institute *of Standards and Technology*) also are characterized as 1-D descriptors.

*17.2.1.2.1 Chemical Formula.* A *chemical formula* is the most common means to represent the chemical structure. Besides being compact and easy to interpret, it conveys the chemical constituents and the number of atoms in a compound. However, it does not give any information on connectivity, stereochemical configuration, or the three-dimensional (3-D) coordinates, which are essential for advanced studies. For example, vasicine (Figure 17.1) has $C_{11}H_{10}N_2O$ as its chemical formula, which only provides details of atom composition and molecular mass.

*17.2.1.2.2 1-D Line Notations.* Linear notation provides the complete constitution and connectivity of chemical compounds as a linear sequence of characters. Among all available 1-D notations, SMILES has found widespread application in the

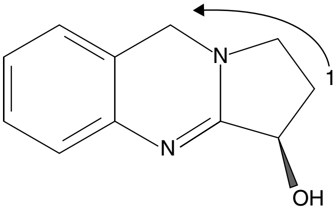


**Figure 17.1** Structure and SMILES format representation of vasicine. SMILES representation can be calculated starting from the position 1 as pointed in the figure.

representation and exchange of chemical information over the Internet. The SMILES format for vasicine is C1CN2CC3=CC=CC=C3N=C2C1O, which can be calculated from Figure 17.1.

### 17.2.2 Two-Dimensional (2-D) Descriptors

The descriptors calculated from two-dimensional (2-D) representations are called *2-D descriptors* and are grouped broadly into three types. Topological indices (Wiener index, molecular connectivity index, and Kappa shape index), 2-D fingerprints (dictionary-based and hash-based), and 2-D fragment descriptors (atom pairs [APs], topological torsions, augmented atoms [AAs], and atom sequence). Typically, the 2-D descriptors are related to the size, rigidity/flexibility, and the electron distribution of the molecule.

***17.2.2.1 Molecular Connectivity or Topological Indices.*** *Topological indices* (TI) are single-value numerical quantities based on certain characteristics of a molecular graph [27, 28] and encode molecular properties such as ring structure, number of atoms, branching, heteroatom content, bond order, and overall shape. They are easy to calculate and hence have found a widespread use among the researchers.

***17.2.2.2 2-D Fingerprint (Binary) Descriptors.*** 2-D fingerprint (binary descriptor) is another commonly used descriptor for chemical database mining [29]. A *fingerprint* is a string of binary characters encoding the information on the presence or absence of substructures in a given molecule. Each binary character is called a bit. These descriptors originally were designed for chemical substructure searching but also have found application in similarity searching [30]. There are two types of binary descriptors described in the literature [31]: dictionary-based (dataset dependent) and hash-based (dataset independent).

*17.2.2.2.1 Dictionary-Based Binary Descriptors.* *Dictionary-based descriptors* are said to be dataset dependent because the reference substructure dictionary contains the fragments from the dataset. For a given molecule x, the appropriate bit is set to 1 (ON in Boolean algebra) when the substructure in x matches the

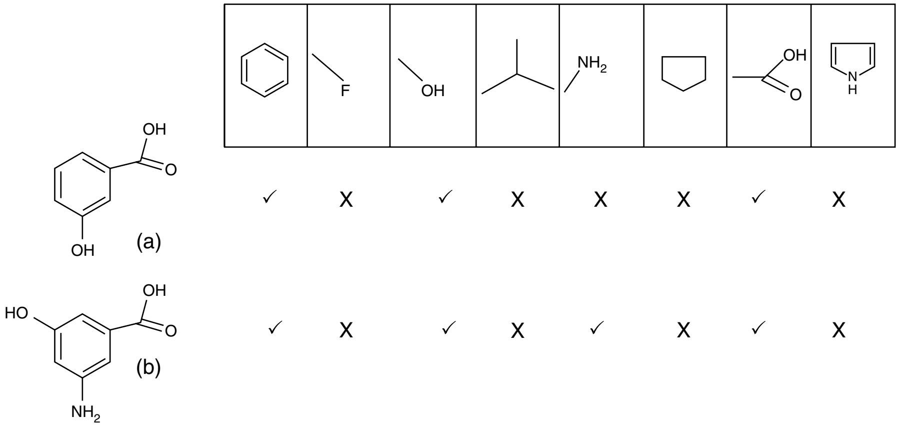


**Figure 17.2** Dictionary-based fingerprint. Molecules A and B are compared against a reference fingerprint dictionary.

fragment present in the dictionary or else it is set to 0 (OFF in Boolean algebra). In Figure 17.2, molecule A comprises three substructures, namely benzene, alcohol, and acid. Therefore, these bits will be set to 1 in the fingerprint.

A limitation of the dictionary-based approach is that every time the dataset changes, a new dictionary based on these changes has to be created for comparison, and the entire fingerprint database has to be recreated. As an example, a dictionary used for DL molecules may not be suitable for comparing with an organometallic compound database. The generation of substructure keys is a rate-limiting step; however, once these keys are generated, the comparison of bitstrings is very rapid using one or more similarity metrics, such as Tanimoto, Treversky, Cosine, or Hamming distance. Of all the similarity metrics proposed, perhaps Tanimoto still remains the most popular coefficient. A list of commonly used metrics is given in Table 17.1, with details of how each metric is calculated. Two examples of dictionary-based fingerprints are the *Barnard Chemical Information* (BCI) fingerprints and the *Molecular Access System* (MACCS) structural keys [8].

*17.2.2.2.2 Hash-Based Binary Descriptors.* To avoid the shortcomings of dictionary-based fingerprints, *hash-based fingerprints* were proposed that eliminate the need for predefined patterns. However, similar to dictionary-based structural keys, these are binary in nature and are designed for database searching applications. The patterns are generated from the molecule itself, and the list of patterns is exhaustive. For example, all possible connected linear paths of the atoms in the molecule ranging from length zero to seven (typically) are considered to be patterns. For pyrrole (shown in Figure 17.3), the patterns of length one are: H1—N2, N2—C3, C3═C4, C4—C5, C5═C6, and C6—N2, whereas the patterns of length two are: H1—N2—C3, N2—C3═C4, C3═C4—C5, C4—C5═C6, C5═C6—N2, C6—N2—H1, and so on. The most commonly used hash-based fingerprints are Daylight and UNITY [8].

**Table 17.1 Most commonly used similarity coefficients with formulas**

| Coefficient | Formula |
|---|---|
| Tanimoto/Jaccard | $c/(a+b-c)$ |
| Tversky | $c/(\alpha a+\beta b+c)$ |
| Dice/ Sorenson/Czekanowski | $2c/(a+b)$ |
| Cosine/Ochiai | $a/\sqrt{}(a.b)$ |
| Simpson | $a/\min(a+b, a+c)$ |
| Hamming/Manhattan/City-block | $a+b-2c$ |

a - Number of bits on in molecule A.
b - Number of bits on in molecule B.
c - Number of bits on in both the molecules A and B.
d - Number of bits off in both the molecules A and B.
$\alpha$,$\beta$ - User-defined coefficients.

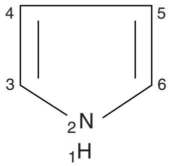


**Figure 17.3** Numbering of pyrrole for showing the hashing scheme. Different patterns of path one to seven can be calculated on the fly.

Recently, Scitegic pipeline pilot *circular fingerprints* [32] have become popular and provide fingerprints with extended connectivity (ECFP) and functional connectivity (FCFP). Initially, each atom is assigned a code based on its properties and connectivity. With every iteration, each atom code is combined with the code of its immediate neighbors to produce the next order code. This process is repeated until the desired number of iterations has been achieved, typically to four iterations, generating ECFP_4 or FCFP_4 fingerprints.

## 17.2.3 Three-Dimensional (3-D) Descriptors

Three-dimensional *(3-D) descriptors* provide information regarding the spatial distribution of atoms and chemical groups in a molecule that leads to the calculation of molecular surfaces and volumes. In nature, molecular recognition occurs in 3-D, and hence, it is intuitive to compare molecules using their 3-D characteristics. 3-D descriptors require conformational properties to be taken into account and hence are more computationally demanding than 1-D and 2-D descriptors.

***17.2.3.1 Pharmacophoric Keys.*** *Pharamacophoric features* in the molecule are atoms or substructures that are important for receptor bonding [33]. There are usually three-point pharamacophoric keys containing three features hydrogen bond donor (HBD), hydrogen bond acceptor (HBA), aromatic rings centers, or hydrophobic centers) or four-point pharamacophoric keys containing four feature combinations. Pharmacophoric keys are widely used 3-D descriptors in molecular similarity and diversity analyses. Binary pharmacophoric keys also can be derived for a molecule based on the presence or absence of a pharmacophore in a molecule. The resulting fingerprints are similar to 2-D fingerprints, but their characteristics are different [34].

***17.2.3.2 Graphs.*** Graph matching algorithms also have been applied to obtain 3-D descriptors [35]. In *3-D graphs*, nodes represent atoms and the interatomic distance between the atoms is represented by edges in the graph. In 3-D graph representations, all nodes are connected to every other node by edges. The maximum common subgraphs (MCS) can be calculated for similarity searching applications using 3-D graph representations. However, the calculation of MCS between two 3-D graphs is more time consuming and computationally intensive than 2-D graphs.

***17.2.3.3 Other 3-D Descriptors.*** Other 3-D descriptors are based on molecular shape and various field representations of a molecule such as steric, electrostatic, or hydrophobic. The superposition of molecular shapes has been used widely as a technique to understand ligand receptor binding and similarity calculations [36]. In these techniques, molecular shape is modeled as the total volume of the compound depicted as spheres [37], Gaussian [38], or other representations of densities such as grid-based encoding [39].

## 17.3 DATABASES

Small molecule databases are essential to characterize novel natural or synthetic compounds and predict their likely biological properties to cut down the biological assays required to determine the properties of a lead compound and minimize toxicity. For several decades, small molecules were the proprietary data of drug companies. Today, large collaborative efforts to annotate the small molecules and analysis software for chemical research [40] have been spearheaded in analogy to the related field bioinformatics. The main publicly available small molecule databases relevant to drug discovery and lead optimization are presented in Table 17.2, with a brief summary of relevant freely available databases.

### 17.3.1 PubChem

*PubChem* [41] is a part of the *National Institute of Health's* (NIH) "Molecular Libraries" initiative and is hosted by the *National Center of Biotechnology* (NCBI). *PubChem* is organized as three linked subdatabases; *PubChem* Substance, *PubChem* Compound, and *PubChem BioAssay*. The *PubChem* Compound databases currently contain >19 million unique structures with computed properties. Searching the database is possible using 1-D and 2-D descriptors described earlier. Each hit provides information about chemical properties, chemical structure, bioactivity, and links to various other related databases such as *PubChem* Substance and *PubMed*.

**Table 17.2** **Summary of relevant publicly available small molecule databases**

| Name | Homepage | Number of compounds | Format# | Data type |
|---|---|---|---|---|
| *PubChem* | pubchem.ncbi.nlm.nih.gov | >19,000,000 | SDF | Small molecule |
| *ChemDB* | cdb.ics.uci.edu/index.htm | >5,000,000 | SMILES, SDF or mol | Small molecules |
| *ChemBank* | chembank.broad.harvard.edu | >800,000 | Text | Small molecules |
| *ChemIDplus* | chem.sis.nlm.nih.gov/chemidplus | >370,000 | mol | Small molecules |
| NCI | cactus.nci.nih.gov/ncidb2 | 260,071 | SDF | Toxicity |
| *SuperToxic* | bioinf-services.charite.de/supertoxic | >60,000 | mol | Toxicity |
| CPDB | epa.gov/NCCT/dsstox/index.html | >1500 | SDF | Toxicity |
| HMDB | hmdb.ca | >6800 | SDF | Human metabolites |
| *DrugBank* | drugbank.ca | >5000 | SDF | Drugs |
| ChEBI | ebi.ac.uk/chebi | 18,186 | mol | Biologically relevant molecules |

#SDF- Structure data file.

### 17.3.2 Chemical Entities of Biological Interest (ChEBI)

Chemical Entities of Biological Interest (ChEBI) [42] is maintained by the European Bioinformatics Institute. It focuses on “small” chemical substances made synthetically or produced naturally, which can affect living organisms. ChEBI relies on two main sources for information: the *Integrated Relational Enzyme* Database (IntEnz) with information on enzyme nomenclature and the Kyoto Encyclopedia of Genes and Genomes Ligand COMPOUND database with information on metabolic compounds and other chemical substances relevant to biological systems. ChEBI also includes an ontological classification in which the relationship between entities and their parent compounds has been specified. A 1-D search can be carried out against ChEBI entries. Furthermore, each ChEBI entry is cross linked to the UniProt protein database enabling access to known biological function of proteins.

### 17.3.3 ChemBank

*ChemBank* [43] is a comprehensive, online informatics system developed through collaboration between the Broad Institute Chemical Biology Program of Harvard and the Massachussetts Institute of Technology (MIT). The aim of the database is to provide biologically relevant pharmacogenomics data and open-source tools to empower the global scientific community in understanding and treating of diseases. The knowledge base stores the high-quality raw screening data and various measurements derived from cells and other biological systems after treatment with small molecules. Several analysis tools that allow studying the relationships among small molecules, cell measurements, and cell states are being developed and updated continuously.

### 17.3.4 ChemIDplus

*ChemIDPlus* [44] is a web-based searchable system that provides access to structural and chemical information for the chemical substances cited in the National Library of Medicine (NLM) databases. Currently, *ChemIDplus* contain more 370,000 compounds that can be searched by their 1-D properties and toxicity indicators such as median lethal dose (LD50) and physicochemical properties like log P, molecular weight, and so on. 2-D similarity and substructure can be performed with user-specified structures.

### 17.3.5 ChemDB

The *ChemDB* [45] is a highly annotated database that supports multiple molecular formats. It is compiled from 150 different sources, which include commercial vendors, catalogs, and publicly available repositories. Currently, the database contains nearly 5 million commercially available small molecules. The chemical data includes predicted or experimentally determined physicochemical properties such as molecular weight, H-bond donors, acceptors, and 3-D structure. The data is available in several formats SMILES (1-D), 2-D graphs, 3-D coordinates (SDF or MOL format), and in fingerprints. The database is searchable by text, SMILES, and physicochemical properties. The tversky coefficient is applied for substructure or similarity searches. *ChemDB* maintains a repository of datasets for machine-learning experiments and also supports online tools for system biology and virtual screening programs.

## 17.4 METHODS AND DATA ANALYSIS ALGORITHMS

The necessity to classify DL substances from ordinary chemicals present in the vast chemical space is of utmost importance [46]. In an effort to select DL molecules, several methods besides quantitative structure activity relationship (QSAR) equations, have been applied to optimize compound selection. Binary classification algorithms such as decision trees and support vector machines frequently are applied to distinguish between DL and non-DL molecules. A variety of methods, ranging from simple count to advanced knowledge-based methods, have been proposed.

### 17.4.1 Simple Count Methods

Simple count methods utilize molecular descriptors that are computed rapidly and are thus highly popular. The most famous of these is Ro5 [16], proposed by Lipinski *et al.* at Pfizer and Vertex. The “Pfizer rule,” or better known as the *“rule of five”* (Ro5), derive guidelines for absorption and permeation of drug molecules and states that a hit (lead) compound has a better chance of survival through the drug discovery pipeline if its molecular weight (MW) is less than 500 (MW $<$ 500), its log $P$ is less than equal to 5 (log $P \leqq 5$), it has five or fewer HBD sites (HBD $\leqq 5$) and a maximum of 10 HBA sites (HBA $\leqq 10$). Ro5 originally was designed as a guideline to pick DL substances, but it has been used as a rule for distinguishing between DL and non-DL compounds. Although Ro5 has dominated drug design, several current drugs do not comply with Lipinski’s rule, with the majority of violations coming form natural products, vitamins, and antibiotics. In one of the studies [47], it was shown that using this criteria, only 66% of the compounds in the *MDL Drug Data Report* (MDDR) database are classified as DL, whereas 75% of the theoretically non-DL compounds from the Available Chemical Directory (ACD), in fact, were regarded as DL.

Similarly, from an analysis by Oprea *et al.* [48], new pharmaceutical substances entering the market exhibit a higher molecular weight and an increased number of hydrogen bond donors and acceptors than is suggested by Ro5, although if a compound fails the Ro5 test, then oral bioavailability problems are likely. At the same time, fulfilling Ro5 does not guarantee that a compound is DL. Moreover, Ro5 does not consider toxicity and structural features, leading to drug failures. Analyses of the failed drugs over past few years have shown that more than 90% of the setbacks are a result of drug toxicity [49, 50].

***17.4.1.1 Example Studies Using Simple Count Methods.*** Ghose *et al.* [51] extended the concept of drug-likeness and carried out a broad analysis of seven different subsets of drugs in the Comprehensive Medicinal Chemistry (CMC) database. These subsets included drugs for the central nervous system, cardiovascular, cancer, inflammation, and infectious disease states molecules. A total of 6304 molecules were included in the study. Based on the calculated physicochemical properties, Ghose *et al.* determined qualifying ranges and preferred ranges that covered 80% and 50% of the database, respectively. Ranges were established for $A\ \log P$ ($-0.4$ to 5.6), molar refractivity (40 to 130), molecular weight (160 to 480), and the number of atoms (20 to 70).

A similar study was conducted by Oprea [52] who performed *Pareto analysis* (a statistical technique that selects a limited number of tasks that account for significant overall effect) on various drug-related databases. Oprea examined the property distribution among several commercial databases including *MACCS-II Drug Data Report* (MDDR), CMC, Current Patent Fast Alert, New Chemical Entities, and ACD databases. The conclusions were that Ro5 does not distinguish between drugs and nondrugs because the distribution of the Ro5 parameters does not differ significantly between drugs and nondrugs. In addition to the Ro5 properties, Oprea also considered counts of ring bonds, rigid bonds, and rotatable bonds in his study, which facilitated the discrimination of drugs and nondrugs.

### 17.4.2 Enhanced Simple Count Methods, Using Structural Features

Counting of substructures also has been employed by many researchers to distinguish between DL and non-DL substances.

***17.4.2.1 Substructure Analysis and Example Studies.*** Substructure analysis usually is performed to distinguish potentially useful compounds [53]. Bemis and Murcko carried out an extensive study of CMC database to characterize molecular skeletons and side chains that occur most frequently in drugs [54, 55].

Muegge *et al.* [56] has proposed a simple selection criteria for DL compounds based on pharmacophore point filters based on the observation that nondrugs often contain fewer functional groups than drugs. Therefore, a minimum count of well-defined pharmacophoric points would distinguish successfully between DL and non-DL compounds. Methods based on functional group filters including this method are significantly less accurate than various machine learning (ML) methods described later. Nevertheless, Muegge *et al.* were able to filter 66%–69% of the MDDR subset and 61%–68% of the CMC subset as DL, whereas only 36% of ACD was found to be DL.

Similarly, Zheng *et al.* [57] developed a chemical space filter by analyzing 50 structural and physicochemical properties on drug (MDDR and CMC) and nondrug databases (ACD). Using molecular saturation-related descriptors and the proportion of heteroatom in a molecule, both of which are molecular-size-dependent descriptors, the developed filter was applied to the *Chinese Natural Product Database* (CNPD). Their filter was reliable, as the entire CNPD was classified DL.

### 17.4.3 ML Methods

During past few decades, several prediction methods have been developed to classify DL and non-DL molecules. Methods dependent on learning algorithms for developing classification rules are called ML methods [58]. Essentially in a ML task, training molecules are represented by *n*-dimensional vectors. These vectors define the point in a chemical space describing the molecule, and an ML algorithm tries to classify the object into distinct groups. In recent years, ML algorithms have gained popularity for following reasons:

- Exponential increase in data generation (biological and chemical)
- Generally ML algorithms perform well in real-world problems
- Well suited for high-dimension data like microarray and images
- Sound mathematical foundations, computationally efficient, and scalability to handle large datasets

ML algorithms are of two types based on supervised and unsupervised learning.

***17.4.3.1 Supervised Classification (SC) Methods.*** In *supervised learning*, we have a prior knowledge, even if only approximate, of the outcome for m samples in the training set. We expect to find a hypothesis h, which closely resembles our prior knowledge for the training set members, and then this hypothesis will serve as a good model for the remaining members of the dataset, usually called the test set. The most common examples of SC methods are discriminant analysis (linear discriminant analysis [LDA] or quadratic discriminant [QDA]), *support vector machines* (SVMs), *artificial neural networks* [ANNs], and classification and regression trees.

*17.4.3.1.1 Neural Networks (NNs).* Neural network [NN]-type algorithms are based on nature inspired architecture. An ANN is an interconnected group of artificial neurons (programming constructs that mimic the properties of biological neurons). The three components of an ANN are the input node, the output node, and the neuron itself. The drawback of this type of network is that they are computationally intensive and the sample size should be large. There are different types of NNs available.

1. *Feed forward neural networks* (FFNNs): These are one of the simplest types of artificial NN, with a unidirectional flow of information. There are no cycles or loops involved, so that information flow is from input nodes to hidden nodes and then to output nodes.
2. *Kohonen Self-Organizing Maps* (SOMs): This is one of the more computationally intensive types of learning algorithm. The basic idea is to reduce the dimensionality without losing useful information and to organize data on the basis of similarity.

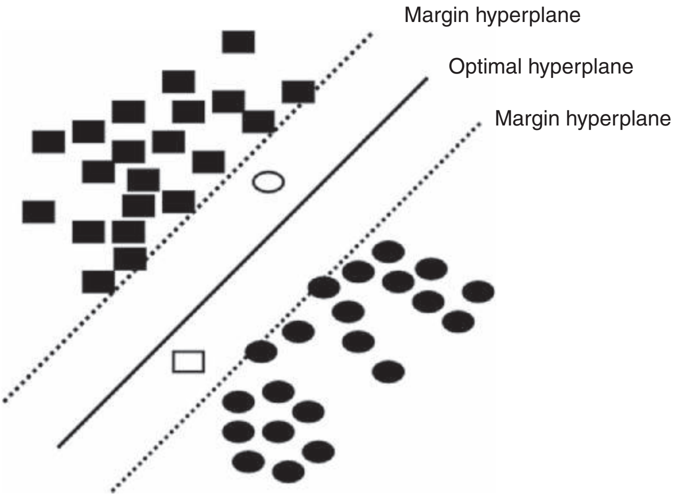


**Figure 17.4** A schematic representation of a binary classification task. Two classes of compounds are separated by calculating an optimal margin hyperplane.

3. *Recurrent NN*: Unlike feed forward NNs, recurrent NNs have a bidirectional dataflow. Although a feed forward NN propagates data linearly, recurrent NN can propagate data from later processing stages to earlier stages.

*17.4.3.1.2 Support Vector Machines (SVMs).* The *support vector machine* [59] is a supervised learning algorithm that recently has received attention in chemoinformatics because of its robust nature and binary classification ability. Several comparison studies have pointed that SVMs almost always outperform other contemporary machine learning techniques and is the best method for classifying molecules [60, 61]. In a typical SVM classification exercise, a training set belonging to two different classes (toxic versus nontoxic or active versus inactive) is projected in a chemical space using various descriptors, and an optimal hyperplane is derived that will best separate the two classes. The basic idea of the SVM principle is easy to interpret and can be understood from Figure 17.4. The disadvantage of using a classifier like SVM is that it does not rank the output, and therefore, it presents a risk of eluding potential ligands that could be improved with minor modifications.

*17.4.3.1.3 k-Nearest Neighbor Classifier (k-NN).* The *nearest neighbour* method is the simplest of the machine learning methods. This method classifies an element by finding its $k$-closest neighbors and by choosing the more common class among the $k$ elements. Usually Euclidian distance is used as the distance metric; however, for text classification, Hamming distance also can be used. The major drawback of $k$-NN technique is that classes with more frequent examples tend to dominate and hence can introduce a bias in prediction. In the past, $k$-NN has been used to classify toxicity data and also has been used in various comparison studies with NN, SVM, and other methods [62].

**17.4.3.2 Unsupervised Classification (UC) Methods.** In *unsupervised learning*, we do not have any prior knowledge of the outcomes. Clustering and partitioning are the two most common examples of an unsupervised learning task.

*17.4.3.2.1 Clustering.* *Clustering* is a technique that tries to organize the variables into relatively homogenous groups called clusters, according to a defined distance measure. The clusters formed exhibit large intracluster similarity and intercluster dissimilarity [63]. Therefore, the result of a cluster analysis would be the number of heterogeneous groups with homogenous contents. One of the methods to choose a diverse set of compounds from a molecular database is to cluster the compounds present in the database and then select a representative compound from each cluster. Clustering techniques are classified into two main categories, hierarchical and nonhierarchical, based on the way the compounds are clustered. *Hierarchical clustering* is the most common clustering approach in which the output is a hierarchy (dendrogram) or a tree. Hierarchical clustering may follow top-down (divisive) or bottom-up (agglomerative) approaches. In the top-down clustering approach, a single large cluster is split down sequentially and recursively into smaller clusters until each object is in its own singleton cluster. Conversely, bottom-up algorithms treat each object as a singleton cluster at the beginning and successfully merge the clusters together into a single cluster that contains all objects. The second class is *nonhierarchal* clustering, and the most popular method of the class is the Jarvis-Patrick Clustering algorithm. It involves the calculation of *k* nearest neighbors of each object in the dataset. Once completed, two molecules are clustered if both of them are on each other's nearest neighbor list and share the *k* minimum number of neighbors between them. The advantage of this approach is its speed, but one problem associated with this methodology is the tendency to produce too many singletons or too few large clusters depending on the clustering criteria.

**17.4.3.3 Example Studies.** Some notable studies include the reports by Ajay *et al.* [19], Sadowski and Kubinyi [64], Wagener and van Geerestein [65], Frimurer *et al.* [47], Takaoka *et al.* [66], and Li and Lai [67]. A summary of the studies carried out by these researchers is provided in Table 17.3.

Ajay *et al.* [19] used a Bayseian neural network (BNN) and a decision tree (c4.5) to build the model for classifying drugs and non-DL molecules. In their study, the training and test sets consisted of 3500 and 2000 compounds each from the CMC and ACD databases, respectively. They used a set of seven one-dimensional and two-dimensional fingerprint descriptors. The results suggest that BNN performed well on all occasions when compared with the decision tree approach. Furthermore, the keys, combined with 1-D descriptors, were the best performing combination than using any single descriptor, with the ability to classify 90% of CMC and ACD compounds and 80% of MDDR compounds correctly.

Table 17.3 Summary of studies conducted in past decade for drug, nondrug classification

| | Methodology | Databases for drugs | Databases for nondrugs | Descriptors | % of drugs correctly predicted | % of nondrugs correctly predicted |
|---|---|---|---|---|---|---|
| Ajay *et al.* [19] | *Bayesian neural network* (BNN) | CMC | ACD | 1-D, 2-D (ISIS Keys) | 90% (CMC) (80%) MDDR | 90% (ACD) |
| Sadowski and Kubinyi [64] | FFNN | WDI | ACD | Ghose and Crippen atom types | 77% (WDI) | 83% (ACD) |
| Wagener and van Geerestein [65] | Decision trees | WDI | ACD | Ghose and Crippen atom types | 92% (WDI) | 66% (ACD) |
| Frimurer *et al.* [47] | FFNN | MDDR | ACD | Concord atom types | 88% | 88% |
| Li and Lai [67] | Probabilistic SVM | WDI | ACD | ECFP Scitegic Pieline Pilot | 93% (WDI) | — |

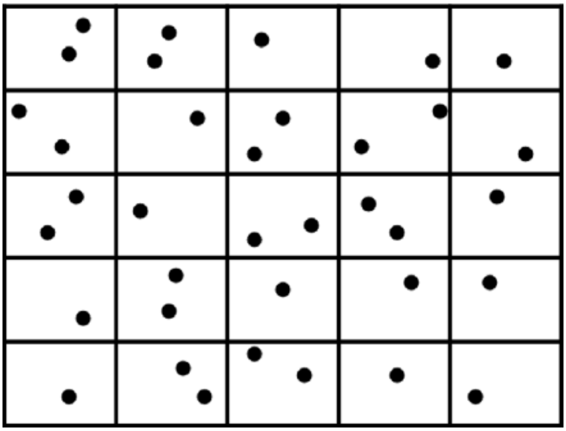


**Figure 17.5** Grid or cell-based selection. Similar molecules tend to fall in similar property space and hence can be partitioned.

Wagener and van Geerestein [65] employed a step-by-step approach using decision tree algorithms. They found that 75% of all drugs can be predicted based on the occurrence of six chemical groups. Likewise, the majority of unsuitable compounds can be ruled out from further analysis based on the presence of specific chemical groups that result in a substance being reactive, toxic, or difficult to synthesize. Sadowski and Kubinyi [64] used Ghose atom types as molecular descriptors, which originally were developed for predicting log P. Their study included ~38,000 DL compounds taken from WDI and ~169,000 non-DL compounds from ACD. A feed forward NN approach was able to classify 83% of the ACD and 77% of the WDI database.

Frimurer *et al.* [47] used the CONCORD atom type descriptors along with a feed forward NN technique to classify DL and non-DL molecules. They were able to classify correctly 88% of DL (MDDR) and non-DL (ACD) compounds. When compared with the rule of five, their method performed significantly better in identifying non-DL compounds.

## 17.5 CONCLUSIONS

As documented in this chapter, chemoinformatics is a rapidly growing field that forms an important link between already established areas such as bioinformatics, computational chemistry and emerging areas like metabolomics, chemogenomics, and pharmacogenomics. Key areas of chemoinformatics like *Molecular similarity* (MS) and *molecular diversity* (MD) analysis are widely accepted concepts in rational drug design. MS lies at the core of all clustering techniques available, whereas molecular diversity is concerned with exploring the structural coverage of the set of molecules in the chemical space. MD lies at the heart of all approaches for compound selection and combinatorial library design. The present level of chemoinformatics understanding has already made it an effective tool in drug discovery and development process.

The current structure and property databases provide a foundation for building *in silico* models with reasonable accuracy. *In silico* models are used for large-scale virtual screening tasks. Although there are various ML techniques available for model building, not all are readily interpretable. There is thus extensive scope for improvement in these methods. Moreover, methods and databases that predict metabolite-likeness are more desirable to those that are restricted to drug-likeness. Furthermore, to ensure the reliability of these models, the inherent redundancy of the datasets should be taken into consideration because redundant datasets can lead to biased results. It is thus important to balance the data such that a particular class of compounds is not overrepresented.

Also, a large proportion of current research effort is directed toward *in silico* modeling of absorption, distribution, metabolism, and excretion properties of leads or potential drug candidates. A variety of models are available for the prediction of physicochemical properties such as log P, oral bioavailability and aqueous solubility, intestinal absorption, and blood-brain barrier penetration. There are some models for metabolism but very few models for excretion and toxicity. Analysis of various drug failure cases has revealed that more than 90% of the setbacks in drug discovery programs are a result of toxicity. Lack of toxicity data could be one of the major bottlenecks in the development of toxicity models. Hence, it is essential that freely available toxic compound databases are developed to get better *in silico* toxicity models. We can expect further development in chemoinformatics approaches to facilitate rational drug design for personalized medicine.

## ACKNOWLEDGMENTS

Varun Khanna acknowledges the award of a Macquarie University Research Scholarship (MQRES).

## REFERENCES

1. F. Brown. Editorial opinion: chemoinformatics - a ten year update. *Curr Opin Drug Discov Dev*, 8(3):298–302, 2005.
2. J.F. Blake. Chemoinformatics - predicting the physicochemical properties of ‘drug-like’ molecules. *Curr Opin Biotechnol*, 11:104–107, 2000.
3. S. Ranganathan. Bioinformatics education - Perspectives and challenges. *PLoS Comput Biol*, 1(6):447–448, 2005.
4. D.R. Flower, I.A. Doytchinova. Immunoinformatics and the prediction of immunogenicity. *Appl Bioinformatics*, 1(4):167–176, 2002.
5. A. Sugden and E. Pennisi. Diversity digitized. *Science*, 289:2305–2305, 2000.
6. E.H. Shortliffe and J.J. Cimino. *Biomedical Informatics: Computer Applications in Health Care and Biomedicine (3rd edition).* Springer, New York, 2006.

7. F. Brown. Chemoinformatics: What is it and how does it impact drug discovery. *Anny Rep Med Chem* 33:375–384, 1998.
8. J. Gasteiger. *Handbook of Chemoinformtics. Form Data to Knowledge*, Volumes 1–4, Wiley-VCH Verlag GmbH, New York, 2003.
9. A.R. Leach and V.J. Gillet. *An Introduction to Chemoinformatics.* Dordrecht, Kluwer Academic Publisher, 2003.
10. D.K. Agrafiotis, D. Bandyopadhyay, J.K. Wegner and H. Vlijmen. Recent advances in chemoinformatics. *J Chem Inform Model*, 47:1279–1293, 2007.
11. M.A. Miller. Chemical database techniques in drug discovery. *Nat Rev Drug Discov*, 1(3):220–227, 2002.
12. M.A. Gallop, R.W. Barrett, W.J. Dower, S.P. Fodor, and E.M. Gordon. Applications of combinatorial technologies to drug discovery. 1. Background and peptide combinatorial libraries. *J Med Chem*, 37(9):1233–1251, 1994.
13. R.S. Bohacek, C. McMartin, and W.C. Guida. The art and practice of structure-based drug design: a molecular modeling perspective. *Med Res Rev*, 16(1):3–50, 1996.
14. T.I. Oprea and H. Matter. Integrating virtual screening in lead discovery. *Curr Opin Chem Biol*, 8(4):349–358, 2004.
15. E. Jacoby. A novel chemogenomics knowledge-based ligand design strategy - Application to G protein-coupled receptors. *Quant Struct-Act Relat*, 20:115–123, 2001.
16. C.A. Lipinski, F. Lombardo, B.W. Dominy, and P.J. Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv Drug Deliv Rev*, 46(1-3):3–26, 2001.
17. W.P. Walters, A. Ajay, and M.A. Murcko. Recognizing molecules with drug-like properties. *Curr Opin Chem Biol*, 3(4):384–387, 1999.
18. T.I. Oprea, A.M. Davis, S.J. Teague, and P.D. Leeson. Is there a difference between leads and drugs? A historical perspective. *J Chem Inform Comput Sci*, 41(5):1308–1315, 2001.
19. A. Ajay, W.P. Walters, and M.A. Murcko. Can we learn to distinguish between "drug-like" and "nondrug-like" molecules? *J Med Chem*, 41(18):3314–3324, 1998.
20. W.L. Jorgensen. The many roles of computation in drug discovery. *Science*, 303(5665):1813–1818, 2004.
21. P. Ertl, S. Roggo, and A. Schuffenhauer. Natural product-likeness score and its application for prioritization of compound libraries. *J Chem Inform Model*, 48(1):68–74, 2008.
22. P.D. Dobson, Y. Patel, and D.B. Kell. 'Metabolite-likeness' as a criterion in the design and selection of pharmaceutical drug libraries. *Drug Discov Today*, 14(1-2):31–40, 2009.
23. S. Gupta and J. Aires-de-Sousa. Comparing the chemical spaces of metabolites and available chemicals: Models of metabolite-likeness. *Mol Divers*, 11(1):23–36, 2007.
24. N. Nikolova and J. Jaworska. Apporaches to measure chemical similarity—a review. *QSAR Comb Sci*, 1006–1026, 2003.
25. B. Cuissart, F. Touffet, B. Cremilleux, R. Bureau, and S. Rault. The maximum common substructure as a molecular depiction in a supervised classification context: Experiments in quantitative structure/biodegradability relationships. *J Chem Inform Comput Sci*, 45(5):1043–1052, 2002.
26. D. Weininger. Smiles, a chemical language and information-system .1. Introduction to methodology and encoding rules. *J Chem Inform Comput Sci*, 28:31–36, 1988.

27. M. Randic. The connectivity index 25 years after. *J Mol Graph Model*, 20(1):19–35, 2001.

28. L.H. Hall and L.B. Kier. Issues in representation of molecular structure the development of molecular connectivity. *J Mol Graph Model*, 20(1):4–18, 2001.

29. P. Willett. Similarity-based virtual screening using 2D fingerprints. *Drug Discov Today*, 11(23-24):1046–1053, 2006.

30. D.K. Agrafiotis. Diversity of Chemical Libraries. *Encyclopedia of Computational Chemistry*, volume 1:A-D: Wiley, New York, 1998, pp 742–761.

31. V.J. Gillet and P. Willett. Compound selection using measures of similarity and dissimilarity, in C. Hansch, editor, *Comprehensive Medicinal Chemistry II*, Elsevier, Atlanta, GA, 2007, pp. 167–192.

32. Pipeline Pilot SciTegic. http://accelrys.com/products/scitegic/.

33. S.D. Pickett, J.S. Mason, and I.M. McLay. Diversity profiling and design using 3D pharmacophores: Pharmacophore-derived queries (PDQ). *J Chem Inform Comput Sci*, 36:1214–1223, 1996.

34. A.C. Good, S.J. Cho, and J.S. Mason. Descriptors you can count on? Normalized and filtered pharmacophore descriptors for virtual screening. *J Comput Aided Mol Des*, 18(7-9):523–527, 2004.

35. J.W. Raymond and P. Willett. Similarity searching in databases of flexible 3D structures using smoothed bounded distance matrices. *J Chem Inf Comput Sci*, 43(3):908–916, 2003.

36. C. Lemmen and T. Lengauer. Computational methods for the structural alignment of molecules. *J Comput Aided Mol Des*, 14(3):215–232, 2000.

37. M. Petitjean. Geometric molecular similarity from volume based distance minimization: Application to saxitoxin and tetrodotoxin. *J Comput Chem*, 16:80–90, 1995.

38. J.A. Grant and B.T. Pickup. A Gaussian description of molecular shape. *J Phys Chem*, 99:3503–3510, 1999.

39. S. Putta, C. Lemmen, P. Beroza, and J. Greene. A novel shape-feature based approach to virtual library screening. *J Chem Inform Comput Sci*, 42:1230–1240, 2002.

40. P. Baldi. Chemoinformatics, drug design, and systems biology. *Genome Inform*, 16:281–285, 2005.

41. C.P. Austin, L.S. Brady, T.R. Insel, and F.S. Collins. NIH Molecular Libraries Initiative. *Science*, 306:1138–1139, 2004.

42. K. Degtyarenko, P. de Matos, M. Ennis, J. Hastings, M. Zbinden, A. McNaught, R. Alcantara, M. Darsow, M. Guedj, and M. Ashburner. ChEBI: A database and ontology for chemical entities of biological interest. *Nucleic Acids Res*, 36:D344–350, 2008.

43. K.P. Seiler, G.A. George, M.P. Happ, N.E. Bodycombe, H.A. Carrinski, S. Norton, S. Brudz, J.P. Sullivan, J. Muhlich, M. Serrano, P. Ferraiolo, N.J. Tolliday, S.L. Schreiber, and P.A. Clemons. ChemBank: a small-molecule screening and cheminformatics resource database. *Nucleic Acids Res*, 36:D351–359, 2008.

44. P.M. Liwanag, V.W. Hudson, and G.F. Hazard. ChemIDplus: An experimental public chemical information and structure search system. *Abstr Paper Am Chem Soc*, 218:U361–U362, 1999.

45. J. Chen, S.J. Swamidass, Y. Dou, J. Bruand, and P. Baldi. ChemDB: a public database of small molecules and related chemoinformatics resources. *Bioinformatics*, 21:4133–4139, 2005.

46. M.C. Hutter. In silico prediction of drug properties. *Curr Med Chem*, 16:189–202, 2009.
47. T.M. Frimurer, R. Bywater, L. Naerum, L.N. Lauritsen, and S. Brunak. Improving the odds in discriminating "drug-like" from "non drug-like" compounds. *J Chem Inform Comput Sci*, 40:1315–1324, 2000.
48. T.I. Oprea, T.K. Allu, D.C. Fara, R.F. Rad, L. Ostopovici, and C.G. Bologa. Lead-like, drug-like or "Pub-like": How different are they? *J Comput Aided Mol Des*, 21:113–119, 2007.
49. D. Schuster, C. Laggner, and T. Langer. Why drugs fail–a study on side effects in new chemical entities. *Curr Pharm Des*, 11:3545–3559, 2005.
50. J. Gut and D. Bagatto. Theragenomic knowledge management for individualised safety of drugs, chemicals, pollutants and dietary ingredients. *Expert Opin Drug Metab Toxicol*, 1:537–554, 2005.
51. A.K. Ghose, V.N. Viswanadhan, and J.J. Wendoloski. A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. 1. A qualitative and quantitative characterization of known drug databases. *J Comb Chem*, 1:55–68, 1999.
52. T.I. Oprea. Property distribution of drug-related chemical databases. *J Comput Aided Mol Des*, 14:251–264, 1999.
53. R.P. Sheridan. Finding multiactivity substructures by mining databases of drug-like compounds. *J Chem Inform Comput Sci*, 43:1037–1050, 2003.
54. G.W. Bemis and M.A. Murcko. The properties of known drugs. 1. Molecular frameworks. *J Med Chem*, 39:2887–2893, 1996.
55. G.W. Bemis and M.A. Murcko. Properties of known drugs. 2. Side chains. *J Med Chem*, 42:5095–5099, 1999.
56. I. Muegge, S.L. Heald, and D. Brittelli. Simple selection criteria for drug-like chemical matter. *J Med Chem*, 44:1841–1846, 2001.
57. S. Zheng, X. Luo, G. Chen, W. Zhu, J. Shen, K. Chen, and H. Jiang. A new rapid and effective chemistry space filter in recognizing a druglike database. *J Chem Inform Model*, 45:856–862, 2005.
58. T.M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, Columbus, OH, 1997.
59. V.N. Vapnik. *The Nature of Statistics Learning Theory (Information Science and Statistics)*. New York, Springer, 1995.
60. V.V. Zernov, K.V. Balakin, A.A. Ivaschenko, N.P. Savchuk, and I.V. Pletnev. Drug discovery using support vector machines. The case studies of drug-likeness, agrochemical-likeness, and enzyme inhibition predictions. *J Chem Inform Comput Sci*, 43:2048–2056, 2003.
61. E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider. Comparison of support vector machine and artificial neural network systems for drug/nondrug classification. *J Chem Inform Comput Sci*, 43:1882–1889, 2003.
62. S.R. Amendolia, G. Cossu, M.L. Ganadu, B. Golosio, G.L. Masala, and G.M. Mura. A comparative study of k-nearest neighbour, support vector machine and multi-layer perceptron for thalassemia screening. *Chemometr Intell Lab Syst*, 69:13–20, 2003.
63. B.S. Everitt. *Cluster Analysis*. London, 1993.
64. J. Sadowski and H. Kubinyi. A scoring scheme for discriminating between drugs and nondrugs. *J Med Chem*, 41:3325–3329, 1998.

65. M. Wagener and V.J. van Geerestein. Potential drugs and nondrugs: Prediction and identification of important structural features. *J Chem Inform Comput Sci*, 40:280–292, 2000.

66. Y. Takaoka, Y. Endo, S. Yamanobe, H. Kakinuma, T. Okubo, Y. Shimazaki, T. Ota, S. Sumiya, and K. Yoshikawa. Development of a method for evaluating drug-likeness and ease of synthesis using a data set in which compounds are assigned scores based on chemists' intuition. *J Chem Inform Comput Sci*, 43:1269–1275, 2003.

67. Q. Li and L. Lai. Prediction of potential drug targets based on simple sequence properties. *BMC Bioinformatics*, 8:353, 2007.

# III

# MOTIF FINDING AND STRUCTURE PREDICTION

# 18

# MOTIF FINDING ALGORITHMS IN BIOLOGICAL SEQUENCES

Tarek El Falah, Mourad Elloumi, and Thierry Lecroq

## 18.1 INTRODUCTION

The *motif finding problem* consists in finding substrings that are more or less conserved in a set of strings. This problem is a fundamental one in both computer science and molecular biology. Indeed, when the concerned strings code biological sequences (*i.e.*, DNA, RNA, and proteins), extracted motifs offer biologists many tracks to explore and help them to deal with challenging problems. Actually, a motif generally represents an expression that characterizes a set of biological sequences [4,13]. It generally codes a substructure and/or a biological function [8]. And hence, on the one hand, a motif can help biologists to learn about the biological functions of biological sequences and, consequently, can help them to understand the mechanisms of the biological processes in which these sequences are involved. On the other hand, a motif common to a set of biological sequences can help biologists to determine common biological ancestors to these biological sequences [14].

Despite many efforts, the motif finding problem remains a challenge for both computer scientists and biologists. Indeed, on the one hand, the general version of this problem is Nondeterministic Polynomial (NP)-hard [11], and on the other hand, our incomplete and fuzzy understanding of several biological mechanisms does not help us to provide good models for this problem.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

In the literature, several versions of the motif finding problem have been identified. In this chapter, we are interested in the following ones:

i. *Planted* $(l, d)$*-Motif Problem* (PMP)
ii. *Extended* $(l, d)$*-Motif Problem* (ExMP)
iii. *Edited Motif Problem* (EdMP)
iv. *Simple Motif Problem* (SMP)

Independently of the version of the motif finding problem, we can group motif finding algorithms into two classes:

i. The first class is made up of *pattern-based algorithms*. By using these algorithms, we try to identify the motif itself.
ii. The second class is made up of *profile-based algorithms*. By using these algorithms, we try to identify not the motif itself but the position of the first character of the motif in each string.

In this chapter, we make a survey of algorithms that address the most studied versions of the motif finding problem, which are PMP, ExMP, EdMP, and SMP.

The rest of this chapter is organized as follows: In Section 18.2, we present some preliminaries. In Sections 18.3–18.6, we survey algorithms for the PMP, the ExMP, the EdMP, and SMP, respectively. Finally, in Section 18.7, we present the conclusion for this chapter.

## 18.2 PRELIMINARIES

Let $A$ be a finite alphabet; a *string* is a concatenation of elements of $A$. The *length* of a string $s$, denoted by $|s|$, is the number of the characters that constitute this string. A portion of a string $s$ that begins at a position $i$ and ends at a position $j$, $1 \leq i \leq j \leq |s|$, is called *substring* of $s$. When $i = 1$ and $1 \leq j \leq |s|$, the corresponding substring is called *prefix* of $s$, and when $1 \leq i \leq |s|$ and $j = |s|$, the corresponding substring is called *suffix* of $s$.

The strings coding DNA, RNA, and proteins are built, respectively, from the alphabets {A, C, G, T}, {A, C, G, U}, and {A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}.

Let $s$ and $s$' be two strings of the same length; the *Hamming distance* between $s$ and $s$', denoted by $H(s, s')$, is the number of the positions where $s$ and $s$' have different characters.

The *edit distance* [6], denoted by $D_{\sigma,\gamma,\delta}(s, s')$, between $s$ and $s$' is the minimum cost of a sequence of *edit operations* (*i.e.*, change of cost $\sigma$, insert of cost $\gamma$, and delete of cost $\delta$) that changes $s$ into $s$'.

## 18.3 THE PLANTED (*l*, *d*)-MOTIF PROBLEM

First, we present a formulation of this problem, and then we present algorithms that address this problem.

### 18.3.1 Formulation

The PMP is defined as follows: Given a set of strings $S = \{S_1, S_2, \ldots, S_n\}$ over an alphabet $A$, $|S_1| = |S_2| = \ldots = |S_n| = L$, two integers $l$ and $d$, $0 \leq d < l < L$, find a string $m$, $|m| = l$, such that for each string $S_i$, $1 \leq i \leq n$, there exists a substring $m_i$ of $S_i$, $|m_i| = l$, with $H(m_i, m) \leq d$ [9,10]. The substring $m$ is called *planted* $(l, d)$-*motif*, and the substrings $m_i$, $1 \leq i \leq n$, are called *planted* $(l, d)$-*variants* of $m$.

Let us note that in other papers [8, 14, 5], the PMP is defined with $H(m_i, m) = d$.

The PMP originates in molecular biology from the need to find *transcription factor-binding* sites in genomic sequences.

### 18.3.2 Algorithms

Among pattern-based algorithms for the PMP, we mention algorithms *Pattern-Branching* [9], Chin and Leung's algorithm [1], PMS1, and PMS2 [10]. Among profile-based algorithms for the PMP, we mention algorithm *ProfileBranching* [9].

***18.3.2.1 PatternBranching.*** Algorithm *PatternBranching* [9] operates as follows: For each different substring $m_0$ in $S$, it constructs iteratively a succession of substrings $m_0$, $m_1$, $m_2$, …, $m_d$, where $m_{j+1} = \mathit{BestNeighbor}(m_j)$ for $0 \leq j \leq d$. At each iteration, it scores the current substring $m_j$, $0 \leq j \leq d$, thanks to the score $H(m_j, S)$ and compares this score with the highest score obtained so far.

Let us denote by $h_q(m_j)$ the set of the substrings of length $l$ that are at a Hamming distance equal to $q$, $1 \leq q \leq d$, from $m_j$. And, for any string $S_i$, $1 \leq i \leq n$, of $S$, let us denote by $h(m_j, S_i)$ the minimum Hamming distance between $m_j$ and any substring of length $l$ of $S_i$. So:

$$H(m_j, S) = \sum_{i=1}^{n} h(m_j, S_i) \tag{18.1}$$

*BestNeighbor*$(m_j)$ is an element $p$ of $h_1(m_j)$ with the lowest distance $H(p, S)$.

Here is a pseudocode of algorithm *PatternBranching*:

**Algorithm 18.1** *PatternBranching* $(S, l, d)$

```
Let $m$ be an arbitrary substring of length $l$
for each different substring $m_0$ in $S$ do
  for $j := 0$ to $d$ do
    if $H(m_j, S) < H(m, S)$ then $m := m_j$ endif
    $m_{j+1} := BestNeighbor(m_j)$;
  endfor
endfor
return $m$
```

In [9], the authors described algorithmic details to speed up this algorithm.

When experimented on strings coding DNA sequences (*i.e.*, with the alphabet $A = \{\mathrm{A, C, G, T}\}$), *PatternBranching* achieves good results in finding subtle motifs and succeeds in finding known biological ones. It would be good to experiment this algorithm on sets of strings built from larger alphabets.

***18.3.2.2 Chin and Leung's.*** In [1], Chin and Leung developed a *voting* algorithm to address the PMP. By adopting this algorithm, we operate as follows: Each substring of length $l$ in the strings of $S$ gives one vote to each of its *variants* (*i.e.*, substrings of length $l$ that are at a Hamming distance equal to at most $d$ from this string), under the restriction that each substring of length $l$ in the strings of $S$ gets at most one vote from each string. Hence, the substrings of length $l$ that have got exactly $n$ votes are considered to be planted $(l, d)$-motifs.

The time complexity of this algorithm is $O(nL((|A| - 1)l)^d)$, and the memory space complexity is $O(L((|A| - 1)l)^d + Ln)$, where $A$ is the alphabet.

***18.3.2.3 PMS1.*** Algorithm PMS1 [10] operates in four steps:

i. During the first step, it extracts the different substrings of length $l$ from the strings of $S$. Let us denote by $C_i$ the set of the different substrings of length $l$ of $S_i$, $1 \leq i \leq n$.
ii. During the second step, for each set $C_i$, $1 \leq i \leq n$, and each element $m$' of $C_i$, it generates all the variants of $m$' (*i.e.*, substrings of length $l$ that are at a Hamming distance equal to at most $d$ from $m$'). Let us denote by $C'_i$ the set of the different variants of the elements of $C_i$.
iii. During the third step, for each set $C'_i$, $1 \leq i \leq n$, it sorts the elements of $C'_i$, and it eliminates the duplicates. Let $L_i$ be the sorted list obtained from $C'_i$.
iv. Finally, during the fourth step, the substrings of length $l$ that are in the intersection of all the lists $L_i$'s, $1 \leq i \leq n$, are considered to be planted $(l, d)$-motifs.

The time complexity of algorithm PMS1 is $O(nL^2l(^l_d)(|A| - 1)^d)$. Furthermore, according to Rajasekaran *et al.* [10], the PMP can be solved in a time $O(nL(^l_d)(|A| - 1)^d \frac{l}{w})$, where $w$ is the length of the computer word. A time $O([nL + L(^l_d)^2(|A| - 1)^{2d}]\frac{l}{w})$ is also achievable.

***18.3.2.4 PMS2.*** Algorithm PMS2 [10] operates in four steps:

i. During the first step, it uses algorithm PMS1, for example, to solve the planted $(d + c, d)$-motif problem for some appropriate value $c$, $(d + c) < l$. Let $R$ be the set of all the planted $(d + c)$-motifs found.
ii. During the second step, it chooses a string $S_j$, $1 \leq j \leq n$, of $S$, and it generates all the variants of all the motifs of $R$ that are in $S_j$ with a Hamming distance equal to at most $d$. For every position $i$, $1 \leq i \leq L$, in $S_j$, let $L_i$ be the list of

the motifs of $R$ that are in $S_j$,with a Hamming distance equal to at most $d$, that start at position $i$.

iii. During the third step, for each position $i$ in $S_j$, let $m$' be the substring of length $l$ of $S_j$ starting at position $i$, $m_1$ be an element of $L_i$, and $m_2$ be an element of $L_{(i+l)-(d+c)}$. If the suffix of length $2(d+c)-l$ of $m_1$ is equal to the prefix of length $2(d+c)-l$ of $m_2$, then it appends the suffix of length $l-(d+c)$ of $m_2$ to $m_1$ to get a substring $m$" of length $l$. If the Hamming distance between $m$' and $m$" is equal to at most $d$, then we include $m$" in the list $L$ of candidate solutions for the planted $(l, d)$-motif problem.

iv. Finally, during the last step, it checks whether any elements of $L$ are correctly planted $(l, d)$-motifs.

The steps *i*, *ii* and *iii* represent the first phase of the algorithm, whereas the last step represents the second phase.

The time complexity of algorithm PMS2 is $O(nL\sum_{i=0}^{d}\binom{d+c}{i}(|A|-1)^i\frac{d+c}{w} + znLl + \sum_{i=1}^{L-l+1}|L_i||A|^{l-(d+c)}l)$, where $z$ is the number of the potential substrings of length $l$ extracted during the first phase and $w$ is the length of the computer word. If $d \leq \lfloor\frac{l}{2}\rfloor$, then this complexity becomes $O(nL\binom{d+c}{d}(|A|-1)^d\frac{d+c}{w} + znLl + \sum_{i=1}^{L-l+1}|L_i||A|^{l-(d+c)}l)$.

Now, let us move to profile-based algorithms that address the PMP.

***18.3.2.5 ProfileBranching.*** This algorithm is proposed by Price *et al.* [9]. It is similar to the *PatternBranching* algorithm [9] but with the following changes:

i. Each different substring $m_0$ in $S$ is converted to a profile $X(m_0)$ as follows: Let $m_0$ be a substring then the profile $X(m_0)$ is defined as being a $|A| \times |m_0|$ matrix $(x_{ij})$, where

   a. A row $i$, $1 \leq i \leq |A|$, represents the $i$th element of the alphabet $A$

   b. A column $j$, $1 \leq j \leq |m_0|$, represents the $j$th character of $m_0$

   c. And $x_{ij} = \frac{1}{2}$ if the $i$th element of the alphabet $A$ is equal to the $j$th character of $m_0$ and $x_{ij} = \frac{1}{6}$, else.

ii. The score $H$ for substrings is replaced by an *entropy score* for profiles. This is done as follows: Given a profile $X = (x_{ij})$ and a substring $m_0 = m_0^1 m_0^2 \ldots m_0^l$, let $e(X, m_0)$ be the log probability of sampling $m_0$ from $X$; that is:

$$e(X, m_0) = \sum_{j=1}^{l} \log(x_{ord(m_0^i)j}) \qquad (18.2)$$

where $ord(m_0^i)$ is the order of $m_0^i$ in the alphabet $A$.

Then, for each string $S_i$, $S_i \in S$, we define $e(X, S_i)$ as follows:

$$e(X, S_i) = \max\{e(X, m_0)|m_0 \text{ is a substring of } S_i\} \qquad (18.3)$$

Hence, the *entropy score* of $X$ is:

$$e(X, S) = \sum_{s_i \in S} e(X, S_i) \tag{18.4}$$

The *relative entropy* of two profiles $X$ and $X$' is defined as follows:

$$re(X', X) = \sum_{i} x'_{ij} \log\left(\frac{x'_{ij}}{x_{ij}}\right) \tag{18.5}$$

iii. The branching method is modified to apply to profiles.

iv. The top scoring profile is used as a seed to the *Expectation Maximization* (EM) algorithm [2].

So, algorithm *ProfileBranching* operates as follows: For each different substring $m_0$ in $S$, it converts this substring to a profile $X(m_0) = X_0$; then it constructs iteratively a succession of profiles $X_0, X_1, X_2, \ldots, X_d$, where $X_{j+1} = BestNeighbor(X_j)$ for $0 \leq j \leq d$. At each iteration, a profile $X_j$, $0 \leq j \leq d$, is scored, thanks to the *entropy score* $e(X_j, S)$, and compared with the top scoring profile $X$ found so far. Finally, it calls the EM algorithm [2] to converge on the top scoring profile $X$ found.

Let us denote by $K_q(X)$ the set of profiles obtained from a profile $X$ by *amplifying* the probabilities of $q$ characters in $q$ components of $X$ to create a new profile $X$' $=$ ($x$'$_{vw}$) with relative entropy equal to $\rho$, where $\rho$ is an implicit parameter.

*BestNeighbor*$(X_j)$ is an element $Y$ in $K_1(X_j)$ with the highest entropy $e(Y, S)$.

Here is a pseudocode of algorithm *ProfileBranching*:

**Algorithm 18.2** *ProfileBranching* $(S, l, d)$

```
Let $X$ be an arbitrary profile
for each different substring $m_0$ in $S$ do
  $X_0 := X(m_0)$
  for $j := 0$ to $d$ do
    if $e(X_j, S) > e(X, S)$ then $X := X_j$ endif
    $X_{j+1} := BestNeighbor(X_j)$
  endfor
endfor
call the EM algorithm with $X$ as a seed.
return the resulting profile
```

Like algorithm *PatternBranching*, algorithm *ProfileBranching* achieves good results in finding subtle motifs in strings coding DNA sequences.

Algorithm *ProfileBranching* runs about five times slower than algorithm *PatternBranching*.

## 18.4 THE EXTENDED (*l*, *d*)-MOTIF PROBLEM

Let us first formulate this problem.

### 18.4.1 Formulation

The ExMP is defined as follows: Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of strings built from an alphabet $A$, $|S_1| = |S_2| = \ldots = |S_n| = L$, and let $l$ and $d$ be two integers, $0 \leq d < l < L$. Find a string $m$ built from the alphabet $A$ such that there exists at least $k$ substrings $m_1, m_2, \ldots, m_k$, $|m_i| = |m|$, $1 \leq i \leq k$, appearing in the strings of $S$ and any substring $m_i$, $1 \leq i \leq k$, differs from $m$ in at most $d$ positions over any window of $l$ characters, $l \leq |m|$ [14, 5]. The substring $m$ is called *extended* ($l$, $d$)-*motif*, and the substrings $m_i$, $1 \leq i \leq k$, are called *extended* ($l$, $d$)-*variants* of $m$.

The ExMP is defined to fix two main weaknesses in the PMP:

i. First, biologists seldom get a set of biological sequences in which each sequence contains a variant of the motif. This is due to experimental errors. They usually get a set of biological sequences; not all of them contain variants.
ii. Second, biologists usually do not know the exact length of the motif. At best, they only know the range for the length.

### 18.4.2 Algorithms

To the best of our knowledge, the only existing algorithms that address the ExMP are pattern-based ones. In this section, we describe two algorithms, Styczynski *et al.*'s algorithm [14] and *exVote* [5].

***18.4.2.1 Styczynski et al.'s.*** Styczynski *et al.*'s algorithm [14] operates in two phases:

i. During the first phase, called *scan phase*, it concatenates the strings of $S$ in a single string $t$, and then it constructs a symmetric matrix $M$ of size $N^2$, where $N = n(L - |m| + 1)$ and $|m|$ is a length supplied by the user, defined as follows: A row $i$ or a column $j$ represents the position in $t$ of a substring of length $|m|$ that is not in overlap between two strings of $S$, and a cell $M[i, j]$ represents the Hamming distance between the substring of length $|m|$ that starts at position $i$ in $t$ and the substring of length $|m|$ that starts at position $j$. So, the matrix $M$ contains the pairwise Hamming distances between the substrings of length $|m|$ found in the set $S$. Actually, the matrix $M$ represents a graph in which a vertex represents a substring of length $|m|$ in the set $S$ and an edge connecting two vertices expresses the fact that the Hamming distance between the corresponding substrings is equal to at most $2d$.
ii. During the second phase, called *convolution phase*, it basically clusters the substrings of length $|m|$, extracted during the scan phase, to make motifs.

The clustering is made thanks to Tomita *et al.*'s optimal algorithm [16] for finding all cliques in the graph represented by the matrix $M$.

Let us note that the whole approach adopted by this algorithm is similar to the one adopted by algorithm *Winnower* [8].

The time complexity of this algorithm is $O((nL)^{k+2.376})$ [5]. Of course, this algorithm is far from being useful in practice. For example, for $L = 600, n = 20, l = 14$, and $d = 4$, this algorithm takes more than three months of computing time [5].

***18.4.2.2 exVote.*** Leung and Chin developed another algorithm, called *exVote*, to address the ExMP [5]. This algorithm is based on the voting approach described in [1] to address the PMP. To address the ExMP, each substring of length $|m|$ in the strings of $S$ gives one vote to each of its *variants* (*i.e.*, substrings of length $|m|$ that differ from this substring in at most $d$ positions over any window of $l$ characters, $l \leq |m|$) under the restriction that each substring of length $|m|$ in the strings of $S$ gets at most one vote from each string. Hence, the substrings of length $|m|$ that have received at least $k$ votes are considered to be extended $(l, d)$-motifs.

Algorithm *exVote* is of complexity $O(nL\sigma(|m|, l, d))$ in computing time, where $\sigma(|m|, l, d)$ is the number of the extended $(l, d)$-variants of a substring of length $|m|$. When $A = \{\mathrm{A, C, G, T}\}$, this number is always less than $|A|^{|m|} = 4^{|m|}$. When $k \approx n$, which is usually much larger than $|m|$, the time complexity becomes then $O(nL|A|^{|m|}) = O(nL4^{|m|})$ and is much smaller than $O((nL)^{k+2.376})$, which is the time complexity of Styczynski *et al.*'s algorithm [14].

Algorithm *exVote* is of complexity $O(|A|^{|m|} + nL)$ in memory space. When $A = \{\mathrm{A, C, G, T}\}$, this complexity becomes $O(4^{|m|} + nL)$, and it would not create much of a problem when $|m|$ is small. To handle large $|m|$, Leung and Chin describe techniques to reduce the memory space complexity without increasing too much the time complexity.

Algorithm *exVote* achieves good results by running faster than Styczynski *et al.*'s algorithm [5].

In [5], a modified version of *exVote* is developed to process the *Further Extended $(l, d)$-Motif Problem* (FExMP).

## 18.5 THE EDITED MOTIF PROBLEM

Let us formulate this problem.

### 18.5.1 Formulation

The EdMP is defined as follows: Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of strings, of average length $L$ and built from an alphabet $A$, and let $l$, $d$, and $q$ be three integers; find all the substrings of length $l$ in $S$, called *edited motifs*, such that each substring has at least $q$ *edited variants* in at least $q$ distinct strings of $S$. A substring $s$ is an *edited variant* of another substring $s$' if $D_{\sigma,\gamma,\delta}(s, s') \leq d$ [12, 11, 15].

### 18.5.2 Algorithms

To the best of our knowledge, the only existing algorithms that address the EdMP are pattern-based ones, they are *Speller* [12], Deterministic Motif Search (DMS) [11], Edit Distance Motif Search (EDMS) EDMS1, and EDMS2 [15]. In this section, we describe only *Speller* [12] and DMS [11].

***18.5.2.1 Speller.*** Algorithm *Speller* [12] operates in two steps:

i. During the first step, called *preprocessing step*, it constructs a *Generalized Suffix Tree* (GST) associated with the strings of $S$. This can be done *via* Ukkonen's online linear algorithm [17].
ii. During the second step, called *spelling step*, for every substring $m$ of length $l$ in the strings of $S$, it searches in the GST for the substrings of length $l$ that are at an edit distance of at most $d$ from $m$. If at least $q$ of the strings of $S$ have each one an edited variant of $m$, then $m$ is an edited motif.

This algorithm is of complexities $O(n^2Ll^d|A|^d)$ in computing time and $O(\frac{n^2L}{w})$ in memory space, where $w$ is the length of the computer word.

***18.5.2.2 DMS.*** Algorithm DMS [11] operates as follows: First, it extracts the different substrings of length $l$ from the $n$ strings of $S$. Then, for each substring $m$ of these substrings, it generates all the edited variants of $m$. This generation is made thanks to Myers's algorithm [7]. Then, for each edited variant of $m$, it determines the list of the strings of $S$ where this edited variant is a substring. Finally, it merges the lists associated with the different edited variants of $m$ into a single list. If the size of this list is at least equal to $q$, then $m$ is an edited motif.

This algorithm is of complexities $O(n^2Ll^d|A|^d)$ in computing time and $O(nLl^d|A|^d)$ in memory space. The memory space complexity can be reduced to $O(nLd + l^d|A|^d)$ [11].

## 18.6 THE SIMPLE MOTIF PROBLEM

Let us first formulate this problem.

### 18.6.1 Formulation

First, let us give some definitions related to this problem: A *simple motif* is a string built from an alphabet $A\cup\{?\}$ that cannot begin or end with "?," where "?" is a *wildcard* character; it can be replaced by any character from $A$. The length of a simple motif is the number of the characters that constitute this motif, including the wildcard characters. The class of the simple motifs in which each simple motif is of length $u$ and has exactly $v$ wildcard characters will be denoted by $(u, v)$-class.

Now let us formulate the SMP: Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of strings built from an alphabet $A$ and $l > 0$ be an integer; find all the simple motifs of length equal to at most $l$ with anywhere from 0 to $\lfloor l|2 \rfloor$ ?'s, and for each simple motif, give the number of times it appears in the strings of $S$ [11].

### 18.6.2 Algorithms

To the best of our knowledge, the only existing algorithms that address the SMP are pattern-based ones. In this section, we present two algorithms, Simple Motif Search (SMS) [11] and *Teiresias* [3]. Actually, algorithm *Teiresias* does not address the SMP but addresses a problem *close* to the SMP.

***18.6.2.1 SMS.*** Algorithm SMS [11] operates as follows:

For each $(u, v)$-class, $0 \leq u \leq l$ and $0 \leq v \leq \lfloor l/2 \rfloor$:

i. First, it extracts all the substrings of length $u$ in the strings of $S$.
ii. Then, it sorts the substrings of length $u$ extracted during the step $i$, only with respect to the nonwildcard positions.
iii. Finally, it scans through the sorted list and counts the number of times each simple motif appears.

The time complexity of this algorithm is $O(l^{l/2}N)$, where $N = \sum_{i=1}^{n} |S_i|$.

In [11], the authors implemented algorithm SMS both sequentially and in parallel. In the sequential implementation on a Pentium 4, 2.4 Ghz machine with 1 GB RAM, SMS takes around 7.25 hours.

***18.6.2.2 Teiresias.*** As we said earlier, algorithm *Teiresias* [3] does not address the SMP but addresses a problem that is *close* to the SMP. So, let us first give some definitions related to this problem:

A *simple motif* has the same definition as in the SMP. A simple motif $m$ is called $< l, d >$*-motif* if every simple motif of $m$ of length at least $l$ contains at least $d$ characters belonging to $A$. An *elementary* $< l, d >$ *motif*-is a substring of length $l$ that contains exactly $d$ characters belonging to $A$. A simple motif $m$' is said to be *more specific* than a simple motif $m$ if $m$' can be obtained from $m$ by changing one or more ?'s of $m$ into characters belonging to $A$ and/or by adding one or more characters belonging to $A \cup \{?\}$ to the extremities of $m$. A simple motif $m$ is said to be *maximal* if there is no simple motif $m$' that is more specific than $m$ and which appears in more strings of $S$ than $m$.

Now let us formulate the problem addressed by algorithm *Teiresias*: Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of strings built from an alphabet $A$ and $l$, $d$, and $q$ be three positive integers; find all the maximal $< l, d >$-motifs in $S$ that occur in at least $q$ distinct strings of $S$.

**Table 18.1 Synoptic table of some motif finding algorithms**

| Problems | Algorithms | Time | Space |
|---|---|---|---|
| PMP | *PatternBranching* [9] | | |
| | *ProfileBranching* [9] | | |
| | PMS1 [10] | $O(nL^2 l \binom{l}{d}(\|A\|-1)^d)$ | |
| | PMS2 [10] | $O\left(nL\sum_{i=0}^{d}\binom{d+c}{i}(\|A\|-1)^i\frac{d+c}{w}+ znLl+\sum_{i=1}^{L-l+1}\|L_i\|\,\|A\|^{l-(d+c)}\,l\right)$ | |
| ExMP | Styczynski *et al.*'s [14] | $O((nL)^{k+2.376})$ | |
| | *ExVote* [5] | $O(nL\sigma(\|m\|, l, d))$ | $O(\|A\|^{\|m\|}+nL)$ |
| EdMP | *Speller* [12] | $O(n^2Ll^d\|A\|^d)$ | $O\left(\frac{n^2L}{w}\right)$ |
| | DMS [10] | $O(n^2Ll^d\|A\|^d)$ | $O(nLd + l^d\|A\|^d)$ |
| | EDMS1, EDMS2 [15] | | |
| SMP | SMS [11] | $O(l^{l/2}N)$ | |
| | *Teiresias* [3] | $\Omega(l^dN\log(N))$ | |

Algorithm *Teiresias* operates in two steps.

i. During the first step, it identifies the elementary $< l, d >$-motifs in the strings of $S$.

ii. Then, during the second step, it superposes overlapping elementary $< l, d >$-motifs identified during the first step to obtain larger $< l, d >$-motifs. The obtained $< l, d >$-motifs in $S$ that are maximal and that occur in at least $q$ distinct strings of $S$ are solutions to the addressed problem.

Algorithm *Teiresias* is of complexity $\Omega(l^dN\log(N))$ in computing time, where $N = \sum_{i=1}^{n}|S_i|$.

Table 18.1 lists algorithms that address motif finding problems.

## 18.7 CONCLUSION

In this chapter, we have surveyed algorithms that address four versions of the *motif finding problem*: The PMP, the ExMP, the EdMP, and the SMP.

Developing more efficient algorithms that address these versions and others is still an ongoing problem.

## REFERENCES

1. F.Y.L. Chin and H.C.M. Leung. Voting algorithm for discovering long motifs. *Proceedings of the Asia-Pacific Bioinformatics Conference*: 261–272, 2005.
2. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J Roy Statl Soc B*, 39(1):1–38, 1977.
3. A. Floratos and I. Rigoutsos. On the time complexity of the TEIRESIAS algorithm. Research Report RC 21161 (94582), IBM T.J. Watson Research Center, 1998.
4. M. Lapidot and Y. Pilpel. Comprehensive quantitative analyses of the effects of promoter sequence elements on mRNA transcription. *Nucleic Acids Res*, 31(13):3824–3828, 2003.
5. H.C.M. Leung and F.Y.L. Chin. An efficient algorithm for the extended $(l, d)$-motif problem, with unknown number of binding sites. *Proceedings of the Fifth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'05)*: 11–18, 2005.
6. V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Cybern Control Theory*, 10(8):707–710, 1966.
7. E.W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4/5):345–374, 1994.
8. P. Pevzner and S.H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*: 269–278, 2000.
9. A. Price, S. Ramabhadran, and P.A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics*, 1(1):1–7, 2003.
10. S. Rajasekaran, S. Balla, and C.-H. Huang. Exact algorithms for planted motif problems. *J Comput Biol*, 12(8):1117–1128, 2005.
11. S. Rajasekaran, S. Balla, C.H. Huang, V. Thapar, M. Gryk, M. Maciejewski, and M. Schiller. High-performance exact algorithms for motif search. *J Clin Monitor Comput*, 19:319–328, 2005.
12. M.F. Sagot. Spelling approximate repeated or common motifs using a suffix tree. *Proceedings of the Theoretical informatics Conference (Latin'98)*: 111–127, 1998.
13. J. Shapiro and D. Brutlag. FoldMiner: Structural motif discovery using an improved superposition algorithm. *Protein Sci*, 13:278–294, 2004.
14. M.P. Styczynski, K.L. Jensen, I. Rigoutsos, and G.N. Stephanopoulos. An extension and novel solution to the $(l, d)$-motif challenge problem. *Genome Informatics*, 15:63–71, 2004.
15. S. Thota, S. Balla, and S. Rajasekaran. Algorithms for motif discovery based on edit distance. Technical Report, BECAT/CSE-TR-07-3, 2007.
16. E. Tomita, A. Tanaka, and H. Takahashi. An optimal algorithm for finding all the cliques. IPSJ Technical Report of SIG algorithms, The University of Electro-Communications: 91–98, 1989.
17. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

# 19

# COMPUTATIONAL CHARACTERIZATION OF REGULATORY REGIONS

Enrique Blanco

## 19.1 THE GENOME REGULATORY LANDSCAPE

Genomes are genetic information repositories on each cell of a living being. Yeast was the first eukaryote that was sequenced more than one decade ago [31]. Since then, the sequence of many other genomes has been published, becoming publicly available in most cases for the worldwide research community. We now can access from our computer the human genome [80, 39] and the sequence, among others, of the fruit fly [3], mouse [40], chicken [38], chimpanzee [70], cow [69], or rice [30, 41] using any of the popular genome browsers [46, 37].

Once the sequence of nucleotides on each chromosome is assembled, one of the initial tasks is to identify the catalogue of biological signals and regions that shape the genome landscape [11]. Genes are units of hereditary information in the organism. In response to variable internal and external conditions, cells increase or decrease the activation of multiple genes, expressing different gene regulatory programs during their lifetime. Protein-coding genes are translated into proteins, which perform diverse biological functions in the organisms (see more about protein-coding genes in [84]). Noncoding genes, such as those that give rise to microRNAs, are responsible for other essential processes in cells (see [29] for a comprehensive gene taxonomy).

Gene transcription initiation is considered to be one important control point in most gene regulatory programs. Multiple actors play specific roles during RNA

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

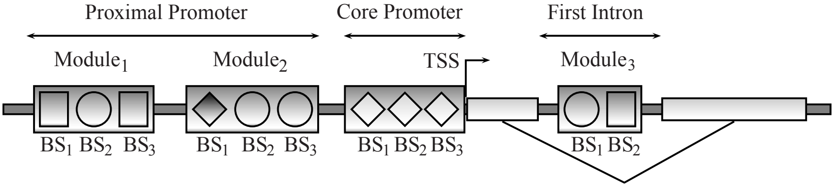


**Figure 19.1** Components of gene regulatory regions.

polymerase II recruitment to initiate the synthesis of transcripts [5]. In eukaryotes, chromatin is packaged into compact structures organized by specific proteins called histones. Nucleosomes, the fundamental packaging units, are histone complexes with DNA wrapping around. In a high regulatory level, gene transcription and nucleosome positioning along the genome are intimately related. Access to gene regulatory regions free of nucleosomes must be granted for the RNA polymerase II to permit the transcription. Multiple chromatin remodelers are thought to imprint different modifications on the histones that constitute the nucleosomes. Such histone marks epigenetically shape long chromosomal regions in the genome to become active or inactive depending on the context (see more about the histone code on [45]). The existence of chromosomal domains that favor coordinated regulation of gene groups results in the nonuniform distribution of genes along the genome [49, 71, 13].

In a low regulatory level, gene transcription is governed locally in the promoter regions. Promoters, as depicted in Figure 19.1, are functional regions inmediately upstream from the transcription start site of genes (TSS) harboring different binding sites for multiple proteins. Transcription factors (TFs) are proteins guiding RNA polymerase II in the recognition of the appropriate inititation site. Core promoter region rules basal expression, whereas specific gene control actually is triggered by a particular collection of TFs in the proximal promoter. Binding sites for transcription factors (TFBSs) are highly variable short sequences (5–15 bp). Protein–protein interactions between TFs, which might confer cooperative or competitive structures such as composites or modules, are poorly known [83]. Although important efforts are being carried out to standardize the construction of libraries [62, 81, 60], current promoter databases still contain incorrect annotations.

Gene expression programs in eukaryotes are highly flexible. Thus, regulatory elements can be found often in regions other than promoters (see Figure 19.1) such as first introns [65] and enhancers [8]. Comprehensive characterization of gene regulatory regions therefore is difficult and complex, making necessary the integration of different computational and experimental techniques to produce accurate descriptions [82]. Using these approaches, several developmental gene regulatory networks have been reconstructed successfully [36, 19]. Thus, substantial improvement in the establishment of the catalogue of functional elements in the human genome was published recently [17]. The availability of gene regulatory architecture maps offers great promises in the elaboration of novel methods to tackle multiple genetic diseases [47].

This chapter briefly reviews the basic computational methods widely employed to characterize regulatory regions, searching for TFBSs in genomic regions that

**Table 19.1 Selected bioinformatics resources to characterize regulatory regions**

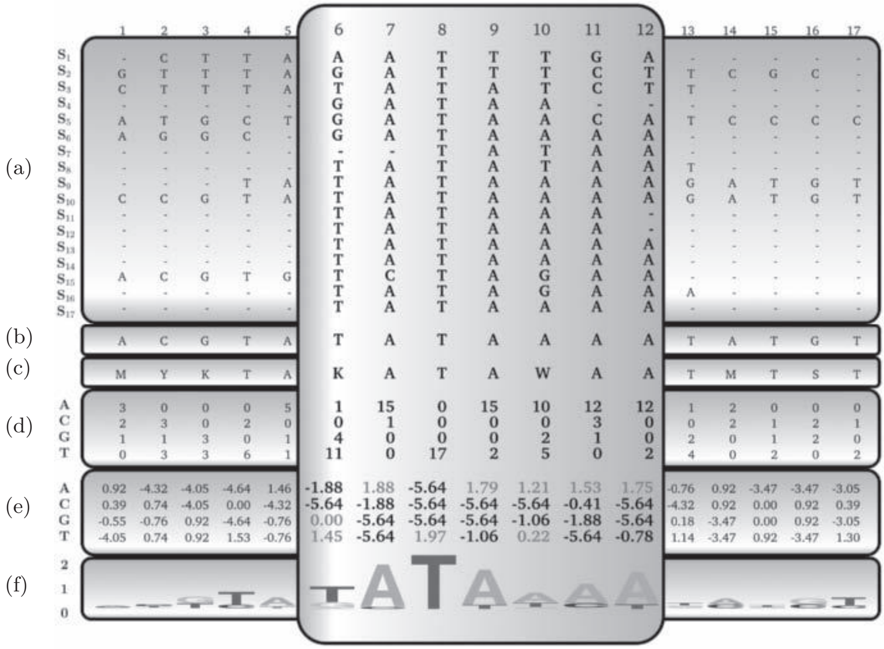
| | | |
|---|---|---|
| Genome browsers | | |
| Ensembl | http://www.ensembl.org | [37] |
| UCSC Genome Browse | http://genome.ucsc.edu | [46] |
| Promoter collections | | |
| EPD | http://www.epd.isb-sib.ch | [60] |
| DBTSS | http://dbtss.hgc.jp | [81] |
| RefSeq | http://www.ncbi.nlm.nih.gov/RefSeq | [62] |
| Regulatory catalogues | | |
| ABS | http://genome.imim.es/datasets/abs2005 | [10] |
| Jaspar | http://jaspar.cgb.ki.se | [15] |
| Oreganno | http://www.oreganno.org | [33] |
| Pazar | http://www.pazar.info | [61] |
| Transfac | http://www.gene-regulation.com | [53] |
| Promoter scanning | | |
| Match | http://www.gene-regulation.com | [43] |
| MatScan | http://genome.imim.es/software/meta | [12] |
| RSAtools | http://rsat.ulb.ac.be/rsat | [78] |
| Phylogenetic footprinting | | |
| Conreal | http://conreal.niob.knaw.nl | [7] |
| eShadow | http://eshadow.dcode.org | [57] |
| Footprinter | http://genome.cs.mcgill.ca/cgi-bin/FootPrinter3.0 | [9] |
| rVISTA | http://rvista.dcode.org | [52] |
| TF-Map alignment | http://genome.imim.es/software/meta/index.html | [12] |
| Motif finding | | |
| Gibbs sampler | http://bayesweb.wadsworth.org/gibbs/gibbs.html | [48] |
| Melina2 | http://melina.hgc.jp | [56] |
| Meme | http://meme.sdsc.edu | [4] |
| Composite prediction | | |
| DIRE | http://dire.dcode.org | [32] |
| Oppossum2 | http://www.cisreg.ca/oPOSSUM | [74] |
| Visualization tools | | |
| gff2ps | http://genome.crg.es/software/gfftools/GFF2PS.html | [2] |
| Pipmaker | http://bio.cse.psu.edu/pipmaker | [68] |
| VISTA | http://pipeline.lbl.gov | [54] |
| Weblogo | http://weblogo.berkeley.edu | [18] |

potentially share similar regulatory mechanisms. First, different representations of these biological signals are introduced. The algorithms that use such models to infer binding sites in other regulatory sequences are described next. Particular focus is given then to learn how to use evolutionary information to filter the initial set of predictions. Finally, experimental methods that currently are used to validate computational predictions are enumerated briefly. Throughout the chapter, several methods to visualize regulatory annotations are presented to the reader. A comprehensive list of selected resources to analyze regulatory sequences is shown in Table 19.1.

## 19.2 QUALITATIVE MODELS OF REGULATORY SIGNALS

Biological signals are short genomic sequences recognized and processed by the cellular machinery that governs the gene expression pathway. TFBSs are regulatory signals located on gene regulatory regions. Because of the variability observed in binding sites of the same TF (see Figure 19.2a), multiple representations have been developed to capture common trends observed on these sequences. The information about a signal class retrieved from input sequences is particularly useful to recognize putative members of this class in other regulatory sequences.

Consensus sequences are the simplest model that can be derived from a multiple sequence alignment of TFBSs (see Figure 19.2b). The consensus can be constructed easily by selecting the nucleotide base more frequently appearing at each position of the signal. The number of matches between the consensus and the candidate sequence can be used to evaluate how similar this site is to the signal class according to the consensus definition. Unless the set of binding sites is uniform, consensus are rather limited predictive models as no information about the variability at each position is modeled [73]. To support some degree of ambiguity on a particular position, the International Union of Pure and Applied Chemistry (IUPAC) extended genetic alphabet of 15 elements allows for special symbols that code for multiple letters (see Table 19.2).



| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (a) | $S_1$ | - | C | T | T | A | A | A | T | T | T | G | A | - | - | - | - | - |
| | $S_2$ | G | T | T | T | A | G | A | T | T | T | C | T | T | C | G | C | - |
| | $S_3$ | C | T | T | T | A | T | A | T | A | T | C | T | T | - | - | - | - |
| | $S_4$ | - | - | - | - | - | G | A | T | A | A | - | - | - | - | - | - | - |
| | $S_5$ | A | T | G | C | T | G | A | T | A | A | C | A | T | C | C | C | C |
| | $S_6$ | A | G | G | C | - | G | A | T | A | A | A | A | - | - | - | - | - |
| | $S_7$ | - | - | - | - | - | - | - | T | A | T | A | A | - | - | - | - | - |
| | $S_8$ | - | - | - | - | - | T | A | T | A | T | A | A | T | - | - | - | - |
| | $S_9$ | - | - | - | T | A | T | A | T | A | A | A | A | G | A | T | G | T |
| | $S_{10}$ | C | C | G | T | A | T | A | T | A | A | A | A | G | A | T | G | T |
| | $S_{11}$ | - | - | - | - | - | T | A | T | A | A | A | - | - | - | - | - | - |
| | $S_{12}$ | - | - | - | - | - | T | A | T | A | A | A | - | - | - | - | - | - |
| | $S_{13}$ | - | - | - | - | - | T | A | T | A | A | A | A | - | - | - | - | - |
| | $S_{14}$ | - | - | - | - | - | T | A | T | A | A | A | A | - | - | - | - | - |
| | $S_{15}$ | A | C | G | T | G | T | C | T | A | G | A | A | - | - | - | - | - |
| | $S_{16}$ | - | - | - | - | - | T | A | T | A | G | A | A | A | - | - | - | - |
| | $S_{17}$ | - | - | - | - | - | T | A | T | A | A | A | A | - | - | - | - | - |
| (b) | | A | C | G | T | A | T | A | T | A | A | A | A | T | A | T | G | T |
| (c) | | M | Y | K | T | A | K | A | T | A | W | A | A | T | M | T | S | T |
| (d) | A | 3 | 0 | 0 | 0 | 5 | 1 | 15 | 0 | 15 | 10 | 12 | 12 | 1 | 2 | 0 | 0 | 0 |
| | C | 2 | 3 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 2 | 1 | 2 | 1 |
| | G | 1 | 1 | 3 | 0 | 1 | 4 | 0 | 0 | 0 | 2 | 1 | 0 | 2 | 0 | 1 | 2 | 0 |
| | T | 0 | 3 | 3 | 6 | 1 | 11 | 0 | 17 | 2 | 5 | 0 | 2 | 4 | 0 | 2 | 0 | 2 |
| (e) | A | 0.92 | -4.32 | -4.05 | -4.64 | 1.46 | -1.88 | 1.88 | -5.64 | 1.79 | 1.21 | 1.53 | 1.75 | -0.76 | 0.92 | -3.47 | -3.47 | -3.05 |
| | C | 0.39 | 0.74 | -4.05 | 0.00 | -4.32 | -5.64 | -1.88 | -5.64 | -5.64 | -5.64 | -0.41 | -5.64 | -4.32 | 0.92 | 0.00 | 0.92 | 0.39 |
| | G | -0.55 | -0.76 | 0.92 | -4.64 | -0.76 | 0.00 | -5.64 | -5.64 | -5.64 | -1.06 | -1.88 | -5.64 | 0.18 | -3.47 | 0.00 | 0.92 | -3.05 |
| | T | -4.05 | 0.74 | 0.92 | 1.53 | -0.76 | 1.45 | -5.64 | 1.97 | -1.06 | 0.22 | -5.64 | -0.78 | 1.14 | -3.47 | 0.92 | -3.47 | 1.30 |

**Figure 19.2** Predictive models of biological signals.

**Table 19.2 IUPAC extended code**

| Symbol | Letters | Meaning |
|---|---|---|
| A | A | Adenine |
| C | C | Cytosine |
| G | G | Guanine |
| T | T | Thymine |
| R | A or G | puRine |
| Y | C or T | pYrimidine |
| M | A or C | aMino |
| K | G or T | Keto |
| S | C or G | Strong interaction (3 H-bonds) |
| W | A or T | Weak interaction (2 H-bonds) |
| B | C or G or T | not A, B follows A |
| D | A or G or T | not C, D follows C |
| H | A or C or T | not G, H follows G |
| V | A or C or G | not T (not U), V follows U |
| N | A or C or G or T | aNy |

The extended genetic code defines, as depicted in Figure 19.2c, a limited set of regular expressions. In general, regular expressions allow for more than one single nucleotide on every position in the regulatory motif. Variability on a position usually depends on the number of times a given nucleotide is observed there in the input dataset (e.g., more than 30% of sequences present this feature). Regular expressions define the motif in terms of subexpressions occurring zero or more times (*) or one or more than one times (+) within the sequence that characterizes the binding sites. The expression (A)*(TCT) + (A)*, for instance, denotes the set of sequences starting by zero or more A symbols, followed by at least one occurrence of the submotif TCT, finishing in zero or more A symbols again. As in consensus sequences, the number of matches between the regular expression and the candidate site determines the score of putative sequence. An overwhelming number of putative sequences can be recognized, however, by a regular expression constructed from a particular set of sites. Many of these combinations might not be present in the original input, though. Because of the lack of quantitative information, consensus and regular expressions are therefore more appropriate models to construct human-readable representations of these sequences [73, 82].

## 19.3 QUANTITATIVE MODELS OF REGULATORY SIGNALS

Popular position weight matrices (PWMs)—also known as position-specific scoring matrices (PSSMs)—are quantitative models that capture the numerical variability on sets of binding sites. The first step to construct a PWM from a multiple sequence alignment of binding sites is producing a position frequency matrix (PFMs). As shown in Figure 19.2d, PFM profiles are two-dimensional arrays of values counting the number of times each nucleotide is observed on a set of functional sites [72, 16].

Normalized PFMs, in which relative instead of absolute frequencies are calculated, are tables of probabilities for each nucleotide along the characteristic motif. Let $F$ be a PFM, where $F(b, i)$ indicates the number of counts of base $b$ in position $i$ of the alignment. The normalized PFM $P$ must be calculated from the frequency distribution as follows:

$$P(b, i) = \frac{F(b, i)}{\sum_{\text{A,C,G,T}} F(b, i)} \tag{19.1}$$

Normalized PFMs represent the composition of a particular class of positive examples of binding sites. To produce PWMs, the initial model is compared with a second description derived from negative examples that improve their discriminative power. Random uniform distributions or background frequencies of each nucleotide on the genome are typical examples of secondary models. Let $Q$ be a table containing the genome composition (*e.g.*, frequency of each base). The PWM $M$ that computes the ratio between positive and negative models is therefore defined as follows:

$$M(b, i) = \frac{P(b, i)}{Q(b)} \tag{19.2}$$

For efficient computational analysis when using these models to predict putative sites, weight matrices must be converted into log-likelihood ratios (see Figure 19.2e). From a given PWM $M$, the log conversion is:

$$(b, i) = \log M(b, i) = \log \frac{P(b, i)}{Q(b)} = \log P(b, i) - \log Q(b) \tag{19.3}$$

Pseudocounts are introduced in the matrix to prevent null values, correcting for small sample sizes [82]. Let $S$ be a table of pseudocounts, the normalized PFM must be then recomputed:

$$P(b, i) = \frac{F(b, i) + S(b)}{\sum_{\text{A,C,G,T}} F(b, i) + S(b)} \tag{19.4}$$

The amount of information available in a weight matrix can be expressed in terms of entropy or in amount of uncertainty. Entropy is measured in bits per symbol for each position of the signal (see [44] for a review of the topic). The uniform random distribution of symbols in a motif results in maximum entropy (*e.g.*, 2 bits in the DNA alphabet). The information content $H(i)$ in position $i$ of the normalized PFM $P$ is defined as [66] follows:

$$H(i) = 2 + \sum_{\text{A,C,G,T}} P(b, i) \log_2 P(b, i) \tag{19.5}$$

Bias in the information content of a given position can be explained in biological terms (*e.g.*, binding energy of the DNA–protein interaction [73]). Most informative positions constitute the core of the matrix (see boxed positions in Figure 19.2). Instead, the context around usually presents weaker conservation levels. Information content can be depicted in a sequence logo, as in Figure 19.2f. The height of each position in the motif is proportional to its information content; the higher the symbol, the more conserved that position is in the motif [66].

PWMs are used to identify new sites that might belong to the same signal class. Weights are used to score every position of the candidate according to its content. Every position within the site is assumed to make an independent contribution to the final score. Under the hypothesis that highly conserved positions are more relevant for the biological activity of the binding site, any site that differs from the profile is scored proportionally to the significance of the mismatching positions in the motif [73]. The final score of a candidate site therefore indicates how similar the prediction is to the profile constructed for this class of TFs. Let $C$ be an input candidate of $l$ nucleotides and $LM$ be a PWM of length $l$ that represents a set of real binding sites. The score $S$ of such a putative site is computed as follows:

$$S(C) = \sum_{i=1}^{l} LM(C(i), i) \tag{19.6}$$

The score $S$ can be normalized into a value $S'$ between 0 and 1 using maximum and minimum matrix scores (Max$S$ and Min$S$, respectively):

$$S'(C) = \frac{S(C) - \text{Min}S}{\text{Max}S - \text{Min}S} \tag{19.7}$$

For sequences longer than the length of a given PWM, a window of this size is slid over them in increments of one position to evaluate each putative binding site (for further information see pattern-driven algorithms). When constructing the matrix, the election of the appropriate length is usually arbitrary. PWMs typically contain the maximum set of consecutive positions that show stronger conservation in comparison to the context in the alignment of binding sites [73].

## 19.4 DETECTION OF DEPENDENCIES IN SEQUENCES

When using weight matrices, the contribution of each nucleotide to the biological signal is supposed to be independent [73]. There are many documented cases, however, in which this assumption is not true [82]. Protein-coding regions, for instance, are constituted of groups of three nucleotides (codons) that are translated into amino acids. Dependencies between neighboring amino acids and bias in the codon usage into proteins produce a nonuniform genomic composition in the coding sequence of genes (see [34] for further details). Oligonucleotide composition in promoter regions may be biased because of certain particular configurations of TFs (see more about promoter identification in [25]). Nevertheless, current biological knowledge about the existence of dependencies between positions in TFBSs is not particularly abundant. To circumvent this, several works have been published on this area (see [1] for a recent review). Oligonucleotide counts, Markov chains, and hidden Markov models are alternative signal representations to model dependencies in these sequences.

Markov chains are the most popular probabilistic model to capture dependencies between consecutive symbols in biological sequences [21]. A Markov chain $M$ basically is defined as a collection of states ($X$), each one denoting a combination of one or more symbols. The probability of a certain symbol following another group of $k$ symbols is defined by the transition probability function ($A$) between two states $a$ and $b$ as follows:

$$A(a, b) = P(x_i = b|x_{i-1} = a) \tag{19.8}$$

The beginning of the sequence is modeled for each state $a$ as follows ($I$, is the initial probability function):

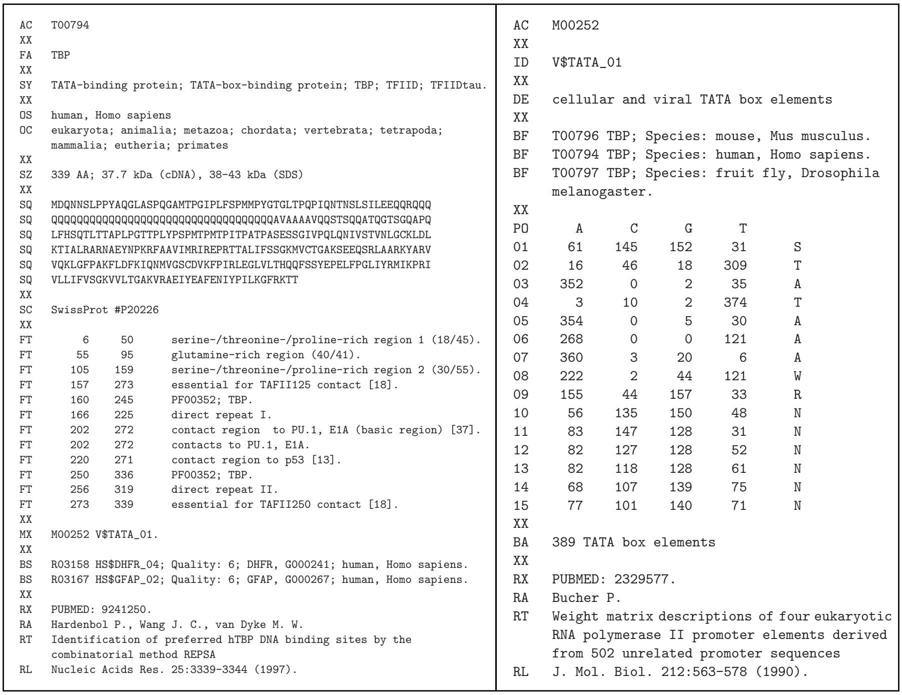
$$I(a) = P(x_1 = a) \tag{19.9}$$

The probability of the input sequence $S = s_1 \ldots s_n$ to follow the model defined by the Markov chain $M$ ($k = 1$) is calculated as follows:

$$\begin{aligned} P(S|M) &= P(x_1 = s_1)P(x_2 = s_2|x_1 = s_1)\ldots P(x_n = s_n|x_{n-1} = s_{n-1}) \\ &= I(s_1)A(s_1, s_2)\ldots A(s_{n-1}, s_n) \\ &= I(s_1)\Pi_{i=2}^{n} A(s_{i-1}, s_i) \end{aligned} \tag{19.10}$$

This value represents the probability that a given sequence has been generated by this model in which dependencies between symbols are incorporated. Similarly to PWMs, positive and negative models are constructed for signal detection (negative models can be calculated from the background frequency in the genome). Therefore, the score in logarithmic terms of a candidate binding site $S$ of $n$ nucleotides using the following predictive model that discriminates between positive and negative signals ($M+$, $M-$) is:

$$\begin{aligned} \log \frac{P(S|M+)}{P(S|M-)} &= \log \frac{I^+(s_1)\Pi_{i=2}^{n} A^+(s_{i-1}, s_i)}{I^-(s_1)\Pi_{i=2}^{n} A^-(s_{i-1}, s_i)} \\ &= (\log I^+(s_1) + \log \sum_{i=2}^{n} A^+(s_{i-1}, s_i)) - \\ &\quad (\log I^-(s_1) + \log \sum_{i=2}^{n} A^-(s_{i-1}, s_i)) \\ &= \log I^+(s_1) - \log I^-(s_1) + \\ &\quad \log \sum_{i=2}^{n} (A^+(s_{i-1}, s_i) - A^-(s_{i-1}, s_i)) \end{aligned} \tag{19.11}$$



```
AC   T00794
XX
FA   TBP
XX
SY   TATA-binding protein; TATA-box-binding protein; TBP; TFIID; TFIIDtau.
XX
OS   human, Homo sapiens
OC   eukaryota; animalia; metazoa; chordata; vertebrata; tetrapoda;
     mammalia; eutheria; primates
XX
SZ   339 AA; 37.7 kDa (cDNA), 38-43 kDa (SDS)
XX
SQ   MDQNNSLPPYAQGLASPQGAMTPGIPLFSPMMPYGTGLTPQPIQNTNSLSILEEQQRQQQ
SQ   QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQAVAAAAVQQSTSQQATQGTSGQAPQ
SQ   LFHSQTLTTAPLPGTTPLYPSPMTPMTPITPATPASESSGIVPQLQNIVSTVNLGCKLDL
SQ   KTIALRARNAEYNPKRFAAVIMRIREPRTTALIFSSGKMVCTGAKSEEQSRLAARKYARV
SQ   VQKLGFPAKFLDFKIQNMVGSCDVKFPIRLEGLVLTHQQFSSYEPELFPGLIYRMIKPRI
SQ   VLLIFVSGKVVLTGAKVRAEIYEAFENIYPILKGFRKTT
XX
SC   SwissProt #P20226
XX
FT        6    50      serine-/threonine-/proline-rich region 1 (18/45).
FT       55    95      glutamine-rich region (40/41).
FT      105   159      serine-/threonine-/proline-rich region 2 (30/55).
FT      157   273      essential for TAFII125 contact [18].
FT      160   245      PF00352; TBP.
FT      166   225      direct repeat I.
FT      202   272      contact region  to PU.1, E1A (basic region) [37].
FT      202   272      contacts to PU.1, E1A.
FT      220   271      contact region to p53 [13].
FT      250   336      PF00352; TBP.
FT      256   319      direct repeat II.
FT      273   339      essential for TAFII250 contact [18].
XX
MX   M00252 V$TATA_01.
XX
BS   R03158 HS$DHFR_04; Quality: 6; DHFR, G000241; human, Homo sapiens.
BS   R03167 HS$GFAP_02; Quality: 6; GFAP, G000267; human, Homo sapiens.
XX
RX   PUBMED: 9241250.
RA   Hardenbol P., Wang J. C., van Dyke M. W.
RT   Identification of preferred hTBP DNA binding sites by the
     combinatorial method REPSA
RL   Nucleic Acids Res. 25:3339-3344 (1997).
```

```
AC   M00252
XX
ID   V$TATA_01
XX
DE   cellular and viral TATA box elements
XX
BF   T00796 TBP; Species: mouse, Mus musculus.
BF   T00794 TBP; Species: human, Homo sapiens.
BF   T00797 TBP; Species: fruit fly, Drosophila
     melanogaster.
XX
PO      A     C     G     T
01     61   145   152    31     S
02     16    46    18   309     T
03    352     0     2    35     A
04      3    10     2   374     T
05    354     0     5    30     A
06    268     0     0   121     A
07    360     3    20     6     A
08    222     2    44   121     W
09    155    44   157    33     R
10     56   135   150    48     N
11     83   147   128    31     N
12     82   127   128    52     N
13     82   118   128    61     N
14     68   107   139    75     N
15     77   101   140    71     N
XX
BA   389 TATA box elements
XX
RX   PUBMED: 2329577.
RA   Bucher P.
RT   Weight matrix descriptions of four eukaryotic
     RNA polymerase II promoter elements derived
     from 502 unrelated promoter sequences
RL   J. Mol. Biol. 212:563-578 (1990).
```

**Figure 19.3** Example of Transfac entries.

Multiple gene finding and promoter applications use this approach to predict biological signals [58]. Markov chains are not appropriate models to process larger sequences, though. Even using a sliding window technique, Markov chains sharply define the overlap between both positive and negative signals around a long region. Hidden Markov models (HMMs) are more sophisticated models in which such boundaries are calculated precisely because of the use of probabilistic methods that compute the optimal path of states when generating the input sequence (see [21] for a comprehensive explanation on HMMs).

## 19.5 REPOSITORIES OF REGULATORY INFORMATION

Several repositories of known transcription regulatory signals have been published (see Table 19.1 for further information). These resources contain information extracted from the literature about experimentally validated TFBSs (see Figure 19.3). Transfac and Jaspar are catalogues of weight matrices constructed for different families of TFs to predict putative binding sites in other sequences (see stats as of June 2009 in Table 19.3). Transfac [53], which appeared more than two decades ago, stores regulatory information classified on diffent TFs, sites, and matrices tables. Jaspar [15] is another popular collection of predictive models derived from experimental

**Table 19.3 Number of entries in different regulatory catalogues**

| Database | TFs | Sites | Matrices | Genes | Reference |
|---|---|---|---|---|---|
| ABS | 68 | 650 | — | 100 | [10] |
| Jaspar | 138 | — | 138 | — | [15] |
| Oreganno | 465 | 14229 | — | 3853 | [33] |
| Transfac | 6133 | 7915 | 398 | 1504 | [53] |

publications that incorporates tools to compare matrices and predict TFBSs in sequences provided by users. ABS consists of a list of experimentally validated sites that are conserved phylogenetically in human and mouse [10]. Accurate information about the location on the genomes of such sites is particularly important to build consistent benchmarks that instruct bioinformatics applications to find binding sites correctly. Oreganno is another comprehensive database of regulatory annotations curated by human experts [33].

Because of the flexiblity of the TFs to recognize binding sites, there is a substantial degree of redundancy in the available regulatory resources. A recent bioinformatics analysis measured the similarity between matrices of several popular collections, reporting the existence of equivalence classes between PWMs of different TFs [67]. This redundancy partially might be caused by the small number of sites usually employed to construct these models [63].

## 19.6 USING PREDICTIVE MODELS TO ANNOTATE SEQUENCES

Successful recognition of regulatory signals is essential for the gene expression machinery in cells. Because of their small size, in comparison with promoter regions, a computational search of signals using predictive models such as those introduced in the previous section is necessary. Pattern-driven algorithms, relying on the use of external repositories of binding sites, are the most important family of computational approaches to characterize the set of binding sites located on promoter regions [14]. The complete protocol to identify TFBSs in regulatory regions is the following:

1. Construction of a catalogue of experimentally annotated sites of a given class
2. Modeling these examples to mask their variability without losing information
3. Detection of new sites in other sequences using the signal models
4. Inspection of predictions to identify modules or composites

To circumvent the complexity of the characterization problem, pattern-driven methods scan unannotated promoter regions simultaneously using repositories for several TFs. This search usually is performed with PWMs from external databases such as Transfac or Jaspar (see [12] for example). A generic version of the pattern-driven algorithm is:

### Algorithm 19.1

**pattern_driven algorithm**

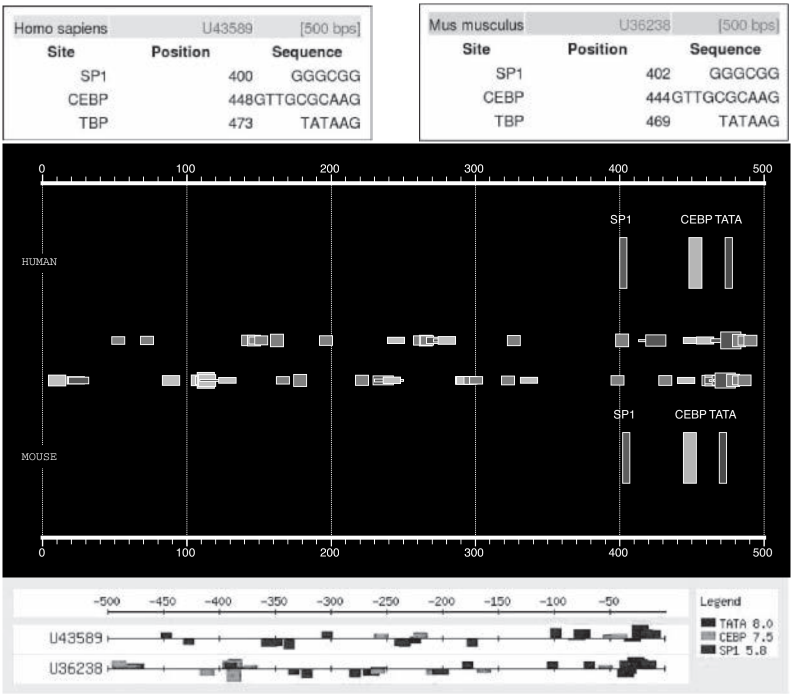
```
{
        INPUT: $S$: sequence; $M$: signal model; $T$: integer;
        OUTPUT: $L$: list of sites;

        $i = 1$;
        (* Apply the model on each window of length $|M|$ *)
        while ($i \leq |S| - |M| + 1$) do
        {
           $j = i + |M|$;
           (* Evaluate the current candidate with this model *)
           score = $M(S_{i,j})$;
           (* Report the candidates above a quality threshold *)
           if (score $\geq T$)
              { AddCandidateList($S_{i,j}$,score,$L$); }
           (* Evaluate next possible binding site *)
           $i = i + 1$;
        }
        ReportCandidates($L$);
        return($L$);
}
```

The computational cost is linear in terms of time being very efficient in the analysis of promoter regions, which are typically 500 to 1000 bp long (depending on the species and the gene in question). Two aspects are critical when running pattern-driven searches: the election of the signal models ($M$) and the threshold to filter out false positives ($T$). The standard procedure to set an appropriate value consists of the evaluation of these predictive models on a set of annotated TFBSs. According to the percentage of true positive examples above a given value, users can set proper stringent thresholds in future searches using these models. As the amount of information on each set of TFBSs is variable, specific threshold values can be associated to every predictive model.

Multiple applications implement variants of the basic pattern-driven strategy. Most programs read sequences in Fasta and output predictions in several standard formats. The Matscan program, which is part of the TF-map alignment suite [12], can process the whole human genome in a few hours using multiple sets of PWMs provided by the user. The output in general feature format (GFF) then can be processed by other graphical applications such as gff2ps [2] to produce high-quality representations of the predictions as shown in Figure 19.4. The RSA tools [78] implement a comprehensive group of promoter characterization techniques, including several pattern-driven programs that produce graphical outputs (see Figure 19.4). Transfac and Jaspar databases provide rudimentary tools to scan promoter regions with their own models of sites as well [53, 15].

Once the map of putative TFBSs is constructed for a given set of gene promoters, accurate inspection of the relationships between predictions of different TF classes (composition and distance of consecutive predictions) may be useful to reveal the existence of composites or regulatory modules [26]. Research on this area is still in



| Homo sapiens | U43589 | [500 bps] |
|---|---|---|
| Site | Position | Sequence |
| SP1 | 400 | GGGCGG |
| CEBP | 448 | GTTGCGCAAG |
| TBP | 473 | TATAAG |

| Mus musculus | U36238 | [500 bps] |
|---|---|---|
| Site | Position | Sequence |
| SP1 | 402 | GGGCGG |
| CEBP | 444 | GTTGCGCAAG |
| TBP | 469 | TATAAG |

**Figure 19.4** Pattern-driven applications.

its infancy, despite several applications recently having produced promising results in particular datasets [32, 74].

## 19.7 COMPARATIVE GENOMICS CHARACTERIZATION

Current methods to analyze a single regulatory region produce a huge number of predictions because of the low specificity of signal models [76]. The availability of many genomes promises, however, substantial improvement in the characterization of gene promoter sequences. Transcription regulation and animal diversity are associated intimately. Emerging evidence suggests that a more sophisticated elaboration of the regulatory mechanisms can be the responsibility of variable organismal complexity along the tree of life [51]. As functional sequences through evolution tend to be more conserved than nonfunctional ones, which might accumulate mutations producing no damage to the organism, interspecies comparisons can be extremely useful to identify common regulatory sequences (see sequence conservation on functional sites in Figure 19.5).

Tagle *et al.* [75] coined the term phylogenetic footprinting to describe the phylogenetic comparisons that reveal evolutionary conserved functional elements in homologous genomic regions. The election of the appropriate species to perform the

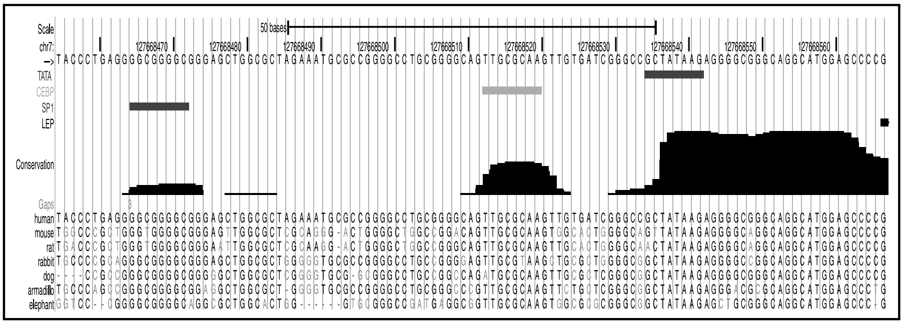


**Figure 19.5** Conservation in regulatory sequences.

comparisons is crucial for every gene in particular as every region of the genome evolves at a different rate [22]. The existence of specific functional elements on each genome also must be taken into account [20] when performing comparative genomics. In the last years, phylogenetic footprinting has become very popular, being accepted as an interesting and efficient method to locate regulatory elements (see [59, 82] for further information).

A distinct family of algorithms named sequence-driven methods approaches the promoter characterization problem from a different perspective [14]. Sequence-driven techniques do not rely on the use of a external dictionary of elements to recognize novel binding sites. Instead, this technique attempts to detect conserved patterns in a set of sequences that are hypothetically coregulated (*e.g.*, orthologous gene promoters or coexpressed genes in microarrays). The protocol to identify TFBSs in regulatory regions using sequence comparisons is as follows

1. Selection of species to perform comparisons
2. Construction of the dataset of orthologous regulatory sequences
3. Sequence comparison to extract the set of evolutionarily conserved regions
4. Analysis of conserved sequences with other predictive models
5. Report candidate binding sites according to their conservation level

There is no particular protocol to implement a comparative approach. A possible sequence-driven algorithm that can be applied systematically to characterize promoter regions might consist of these steps:

## Algorithm 19.2

```
sequence_driven algorithm
{
      INPUT:  S = $S_1 \dots S_n$: list of sequences;
              M = $M_1 \dots M_n$: list of signal models;
              T: integer;
      OUTPUT: L: list of sites;
```

```
        (* Perform the sequence comparison *)
        A =Alignment($S_1 \ldots S_n$);
        (* Extract the conserved regions *)
        R =ExtractRegions(S, A);
        (* Analyze regions with predictive models *)
        L =AnalyzeRegions(R, M, T);
        (* Report the candidates *)
        ReportCandidates(L);
        return(L);
}
```

As multiple combinations of alignment methods and predictive models are possible to analyze regulatory regions, the computational cost of sequence-driven methods is highly variable. Conserved regions and multiple genome comparisons, however, usually are precomputed in most popular genome browsers (*e.g.*, conservation tracks in University of California—Santa Cruz (UCSC) Genome Browser [46] in Figure 19.5).

## 19.8 SEQUENCE COMPARISONS

Sequences are symbolic representations of biological molecules encoding relevant information about their structure, function, and evolution. Sequence comparisons, which reveal the fraction that is similar, are one of the most important tools in molecular biology. Strong sequence similarity usually is assumed to be a good marker for common biological function conserved through evolution. As the number of alignments between two or more sequences is very high, these comparisons must be approached systematically using computational alignment methods. Phylogenetic analyses to identify conserved biological features between distant species usually are conducted using genome-wide alignment methods. These algorithms, basically identify local similarity regions between two genomes, using them as anchors to align the interleaving regions (see [79] for a review).

Several applications produce graphical annotations of the genome alignments (see Table 19.1). VISTA (visualization tool for alignment [54]), for instance, represents conservation scores along sequences as peaks indicating with different colors whether the region is overlapping annotated genes or noncoding regions. The *myc* gene in human and the comparison with orthologous forms in other vertebrates is shown in Figure 19.6. The shadowed areas in the promoter of the gene are interesting regions for further analysis. Using powerful genome-wide alignment methods, the presence of ultraconserved elements in the genomes was reported recently. Such consistent motifs might be playing important regulatory roles [6, 24].

Once the regulatory regions conserved throughout evolution are identified, motif finding programs highlight common patterns on those locations. Pattern discovery is a complex computational problem that cannot be solved optimally. Multiple alternatives have been proposed in the last years to find overrepresented regulatory motifs

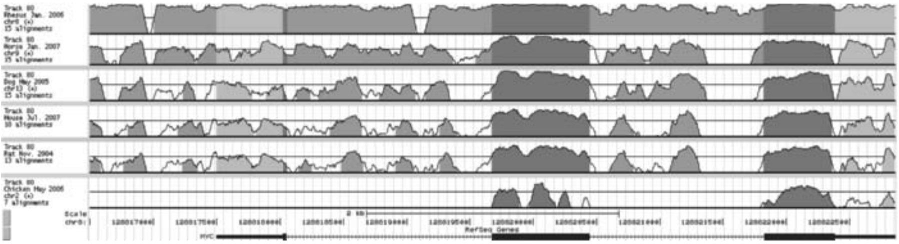


**Figure 19.6** Visualization of genomic regions in VISTA.

in unaligned sequences [56]. Most programs consist of iterative searches of common words, relying on different heuristics to simulate the motif distribution along the input sequences [14, 79]. Phylogenetic information can be incorporated into the motif-finding algorithm to weight the relevance of the occurrences on each species [9].

Expectation maximization (EM) is one of the most popular variants of pattern discovery techniques [4]. EM algorithms estimate the parameters of a probabilistic model that might generate the input dataset of sequences. Such a model contains two components: the occurrences of the motif on each sequence and the remaining positions (background). A fitness function must be defined to assess the quality of the model usually measuring how different the current motif is from the background. EM methods perform an iterative search that refines the initial model until no improvement is observed in the fitness function. The motif is constructed initially from random locations on each sequence. Several initial random points therefore must be evaluated to prevent the fitness function to fall into a local maximum peak. EM algorithms to search motifs of $w$ nucleotides in genomics sequences are structured into the following steps:

## Algorithm 19.3

```
pattern_discovery algorithm
  {
        INPUT:  S = $S_1 \ldots S_n$: list of sequences; $T$: integer;
        OUTPUT: M: signal model;
                L: list of sites;

        (* Choose random locations on each sequence *)
        (* to build the initial model *)
        InitializeModel($M$,$S$);

        i=1;
        convergence = FALSE;
        while (i $\leq$ MAXITERATIONS and convergence is FALSE) do
        {
           for each $S_i$ in S do
           {
              (* E-step: evaluate all candidates of length $w$ *)
              (* sliding a window along the sequence $S_i$ *)
```

```
                Scores = EvaluateCandidates($S_i$,$M$);
                (* M-step: update current model *)
                (* to detect overrepresentations *)
                $M'$ = UpdateModel($M$,Scores);
            }
            if (fitness(M') $\leq$ fitness(M))
            { convergence=TRUE; }
            else
            { M = M'; i=i+1; }
        }

        (* Identify occurences of this model in S *)
        $L$ = pattern_driven(S,M,T);
        ReportCandidates(L);
        return(L);
}
```

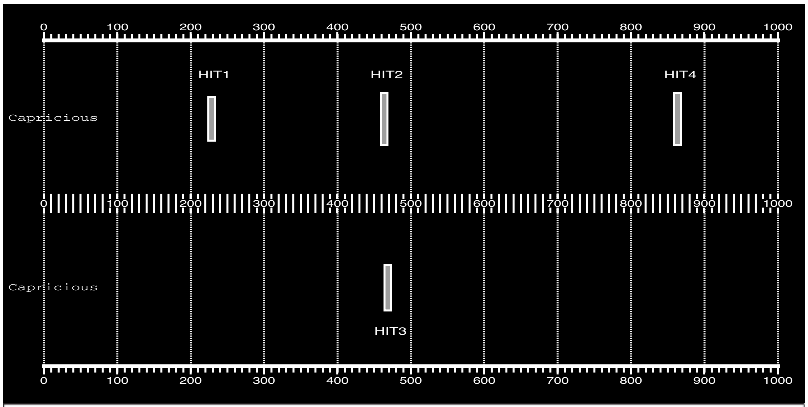
The model M constructed during the iterative procedure can be used to identify putative sites in the input sequences. Optimal motif width can be estimated prior to running the algorithm in a range determined by the user (between 5 and 15 bp for TFBSs) [4].

## 19.9 COMBINING MOTIFS AND ALIGNMENTS

A substantial reduction in the number of predictions can be achieved when phylogenetic footprinting is combined with predictive models. Searches of binding sites using catalogues of regulatory information typically produce too many false positives [82]. However, when the search is confined to those regions reported to be conserved evolutionarily, a dramatic reduction of candidate TFBSs is observed (see Figures 19.4 and 19.7). Automatic annotation pipelines can perform binding sites prediction with PWMs on conserved genomic regions [50, 52]. Manual inspection of results still is recommended, though, to identify accurately regulatory sites for further experimental validation [55]. As shown in Figure 19.7, initial predictions obtained using PWMs for a given TF on a particular promoter can be combined with motif finding on phylogenetically conserved regions in other species to filter candidate binding sites that must be validated in the wet lab.

Similar sequences tend to play similar functions. The opposite, however, is not necessarily true. Often similar functions are encoded in higher order sequence elements, and the relation between these and the underlying primary sequence may not be univocal. As a result, similar functions are encoded frequently by diverse sequences. For instance, similar TFBSs can exhibit great variability at the sequence level in different species. Consequently, promoter regions of genes with similar expression patterns may not show sequence similarity, even though they may be regulated by similar configurations of TFs. Several applications have been published recently to overcome this inconvenience [12, 7, 35]. By representing a promoter region in a new alphabet in which the different symbols denote binding sites for different TFs, promoter comparisons can be performed in the appropriate level. For

```
capricious MatScan Gfi 224 233 0.63 + . # AAAATCAGAT
capricious MatScan Gfi 459 468 0.77 + . # AAAATCAAAG  (*)
capricious MatScan Gfi 464 473 0.66 - . # AAAATCTTTG
capricious MatScan Gfi 859 868 0.76 + . # ACAATCACAC
```



```
D.melanogaster 438 TGGGGGGAGATAGGACTAATGGAAAATCAAAGATTTTCTT 478
D.simulans     413 AATGGAGAGATTGGACTAATGGAAAATCAAAGATTTTCTT 453
D.sechellia    414 AATGTAGAGATAGGACTAATGGAAAATCAAAGATTTTCTT 454
D.persimilis   926 AAGCATTTTCGAACTCTAATGGAAAATCAATTTAAAGCCC 966
D.virilis      546 AAGCAATTTTGAACTCTAATGGAAAATCAATTTAAAGCGC 586
                                         ********
```

**Figure 19.7** Improving initial predictions using comparative genomics.

instance, the alignment of MMP13 promoter regions shown in Figure 19.8 does not detect accurately the four TFBSs that have been validated experimentally according to the literature. Instead, the alternative multiple TF map alignment, which is based on comparisons using the TF alphabet, can recover the four sites in the same promoters [12].

The following algorithm is a simplified version of the TF map alignment as introduced in [12]. For every consecutive pair of TF matches in the TF-map alignment, $\lambda$ and $\mu$ penalties are evaluated to weight how many input binding sites are not included in the current alignment and how similar the location of matches on each original sequence is (positional conservation is assumed to denote biological function):

## Algorithm 19.4

**TFmap_pairwise_alignment algorithm**

```
{
      INPUT:  S = $S_1, S_2$: list of sequences;
              M = $M_1 \ldots M_n$: list of signal models;
              T: integer;
```

```
        OUTPUT:
                A: TF-map alignment of sites;

        (* Construct the TF-maps using a catalogue of models *)
        Map1 = pattern_driven($S_1$, M, T);
        Map2 = pattern_driven($S_2$, M, T);
        (* Visit each site on both maps to find TF label matches *)
        for each site $s_i$ in Map1 do
        {
           for each site $s_j$ in Map2 do
           {
              if (TFlabel($s_i$) = TFlabel($s_j$))
              {
                 score = 0.0;
                 for each previous match ($s_{i'}, s_{j'}$) in $A$ do
                 {
                    penalty$_1$=compute$\lambda$($s_i, s_j, s_{i'}, s_{j'}$);
                    penalty$_2$=compute$\mu$($s_i, s_j, s_{i'}, s_{j'}$);
                    score' = computeSimilarity(penalty$_1$,penalty$_2$);
                    if (score' > score)
                    { score = score'; }
                 }
                 registerMatch($s_i, s_j$, A,score);
              }
           }
        }

        ReportOptimalAlignment(A);
        return(A);
}
```

## 19.10 EXPERIMENTAL VALIDATION

Once bioinformatics analysis identifies a set of solid predictions that characterize gene promoter regions, additional biological validation is necessary to confirm the location of functional TFBSs. In contrast to computational biology, experimental validation of targets is expensive and time consuming. Candidates to be verified, therefore, must be supported convincingly by different sources of evidence (predictive models, motif finding, and evolutionary conservation). Numerous experimental techniques can validate whether a binding site is bound by a given protein. The election of the appropriate method depends on many variables: organism, tissue, gene, or TF, in particular. In addition, some experiments are performed *in vivo*, whereas others only can validate *in vitro* binding between protein and DNA. Confirmation of predicted TFBSs in the wet lab can be performed through the following methods (see [23] for a comprehensive review on this topic):

- DNA footprinting is a method to identify approximately the binding site of a given TF on a fragment of genomic sequence [27]. The DNA molecule must

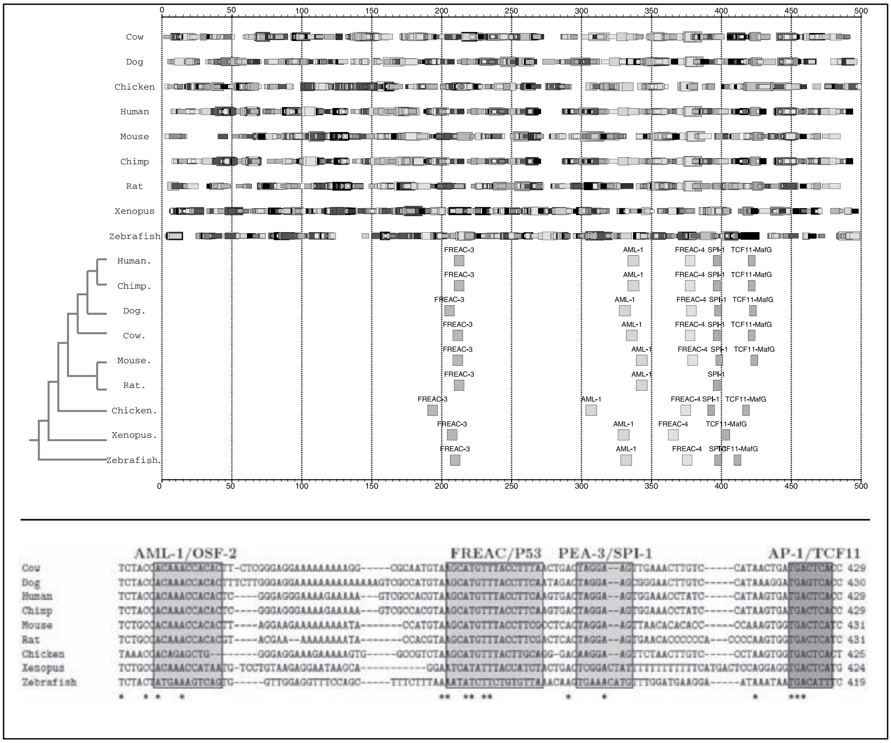


**Figure 19.8** Multiple TF map and sequence alignments.

be labeled radioactively at one end. It then is cleaved with a nuclease that makes random single-stranded cuts. Once the DNA is denatured to separate the two strands, the fragments from the labeled strand are separated on a gel electrophoresis and detected by autoradiography. The pattern of bands from the DNA molecules that were cut in presence of the DNA-binding protein is compared with that of the same fragment without the TF. The part of the genomic sequence bound by the protein is protected from cleavage so that the labeled fragments overlapping the binding site will be missing, leaving a gap or footprint in the gel (see Figure 19.9a).

- Electrophoretic mobility shift assay (EMSA) or gel mobility shift electrophoresis is a common technique to determine whether one protein complex is capable of binding to a given DNA sequence [28]. EMSA experiments are based on the effect of a bound protein on the migration of DNA in an electric field. When analyzed by gel electrophoresis, DNA molecules bound by TFs will move more slowly through the gel than naked genomic sequences. Initially, a short DNA fragment of specific length and sequence is labeled radioactively and mixed with a cell extract. The mixture then is loaded onto a polyacrylamide gel for running electrophoresis. When several sequence-specific proteins bind on a

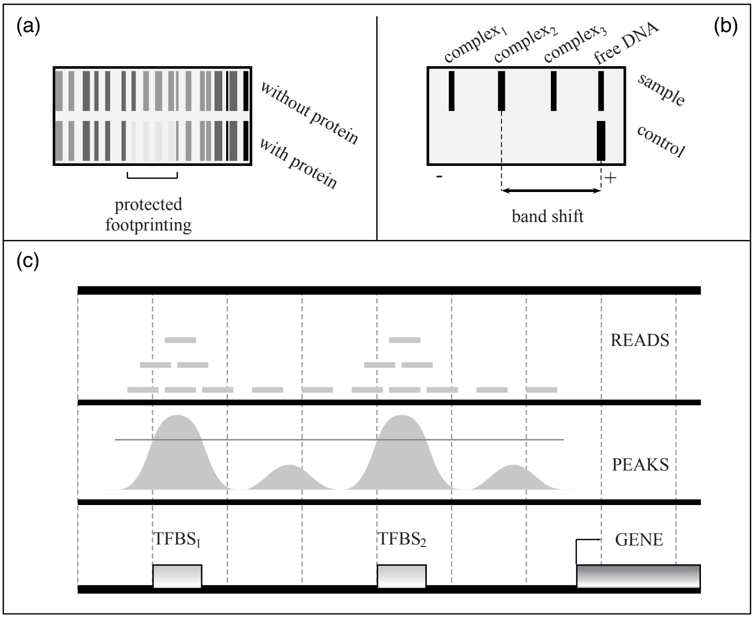


**Figure 19.9** Experimental methods to validate predictions.

region, autoradiography reveals a series of DNA bands, each one with a specific delay resulting from different protein complexes as shown in Figure 19.9b. The precise binding site must be identified by competition using different oligonucleotides on additional experiments.

- Chromatin immunoprecipitation (ChIP) or precipitation of an antigen out of a solution using an antibody is a method used to determine which proteins bind to a particular region on the chromatin of living cells [23]. First, proteins in contact with the DNA are cross-linked to the chromatin and immobilized using formaldehyde fixation. Then, chromatin is fragmented by sonication, and whole protein-DNA complexes are immunoprecipitated using an antibody specific for the protein in question. Finally, DNA molecules from such complexes can be purified and their sequence can be determined by polymerase chain reaction (PCR).
- Systematic evolution of ligands by exponential enrichment (SELEX) is a combinatorial technique to produce oligonucleotides that specifically bind to a target ligand [77]. First, a very large oligonucleotide library, which theoretically contains all possible sequences of $n$ nucleotides, is synthesized. The genomic library is exposed next to the target ligand to remove those elements that do not bind the protein by affinity chromatography. The remaining oligonucleotides are amplified by reverse transcription PCR (RT-PCR) to prepare for subsequent rounds of selection in which the stringency of the elution conditions is increased to identify the tightest-binding sequences.

- ChIP-on-chip (ChIP-chip) is a high-throughput technology that combines chromatin immunoprecipitation with microarrays [64]. DNA microarrays are arrayed devices divided into series of spots, each one containing a genomic fragment of interest that hybridizes to a cDNA sample under certain conditions. Tiling microarrays, in which the whole genome is spotted, provide high-resolution genome-wide maps. ChIP-on-chip devices are microarrays in which the protein – DNA immunoprecipitated fragments are hybridized to determine their location in the genome.
- ChIP-Sequencing (ChIP-Seq) is a recent high-throughput technology that combines chromatin immunoprecipitation with ultra-sequencing machines [42]. Once ChIP fragments are obtained, genome sequencers drive massive sequencing of these short sequences in parallel. Next, computational analysis is required to map each read on the whole genome as shown in Figure 19.9c. The precise map of binding sites for a given TF therefore can be reconstructed with very high resolution.

## 19.11 SUMMARY

Computational identification of transcription regulatory elements is key to understand the gene expression machinery. Multiple bioinformatics applications are available for users to characterize binding sites to transcription factors in gene promoter regions. The sequence of regulatory signals is highly variable. Different approaches to model them have been designed, weight matrices being the most popular ones in the research community. Promoter regions initially are characterized using these predictive models, which provide a preliminary annotation of TFBSs. Performing comparative genomics, most predictions can be filtered out to define a solid set of putative binding sites that are evolutionary conserved. The combination of predictive models, sequence comparisons, and emerging high-throughput expression techniques promises great improvements in the elucidation of gene regulatory networks.

## REFERENCES

1. T. Abeel, Y. Van de Peer, and Y. Saeys. Toward a gold standard for promoter prediction evaluation. *Bioinformatics*, 25:i313–i320, 2009.
2. J. F. Abril and R. Guigó. gff2ps: visualizing genomic annotations. *Bioinformatics*, 8:743–744, 2000.
3. M.D. Adams, S.E. Celniker, R.A. Holt, C.A. Evans, J.D. Gocayne, P.G. Amanatides, S.E. Scherer, P.W. Li, R.A. Hoskins, R.F. Galle, R.A. George, S.E. Lewis, S. Richards, M. Ashburner, S.N. Henderson, G.G. Sutton, J.R. Wortman, M.D. Yandell, Q. Zhang, L.X. Chen, R.C. Brandon, Y.H. Rogers, R.G. Blazej, M. Champe, B.D. Pfeiffer, K.H. Wan, C. Doyle, E.G. Baxter, G. Helt, C.R. Nelson, G.L. Gabor, J.F. Abril, A. Agbayani, H.J. An, C. Andrews-Pfannkoch, D. Baldwin, R.M. Ballew, A. Basu, J. Baxendale, L. Bayraktaroglu, E.M. Beasley, K.Y. Beeson, P.V. Benos, B.P. Berman, D. Bhandari, S. Bolshakov, D. Borkova, M.R. Botchan, J. Bouck, P. Brokstein, P. Brottier, K.C. Burtis, D.A. Busam,

H. Butler, E. Cadieu, A. Center, I. Chandra, J.M. Cherry, S. Cawley, C. Dahlke, L.B. Davenport, P. Davies, B. de Pablos, A. Delcher, Z. Deng, A.D. Mays, I. Dew, S.M. Dietz, K. Dodson, L.E. Doup, M. Downes, S. Dugan-Rocha, B.C. Dunkov, P. Dunn, K.J. Durbin, C.C. Evangelista, C. Ferraz, S. Ferriera, W. Fleischmann, C. Fosler, A.E. Gabrielian, N.S. Garg, W.M. Gelbart, K. Glasser, A. Glodek, F. Gong, J.H. Gorrell, Z. Gu, P. Guan, M. Harris, N.L. Harris, D. Harvey, T.J. Heiman, J.R. Hernandez, J. Houck, D. Hostin, K.A. Houston, T.J. Howland, M.H. Wei, C. Ibegwam, M. Jalali, F. Kalush, G.H. Karpen, Z. Ke, J.A. Kennison, K.A. Ketchum, B.E. Kimmel, C.D. Kodira, C. Kraft, S. Kravitz, D. Kulp, Z. Lai, P. Lasko, Y. Lei, A.A. Levitsky, J. Li, Z. Li, Y. Liang, , X. Lin, X. Liu, B. Mattei, T.C. McIntosh, M.P. McLeod, D. McPherson, G. Merkulov, N.V. Milshina, C. Mobarry, J. Morris, A. Moshrefi, S.M. Mount, M. Moy, B. Murphy, L. Murphy, D.M. Muzny, D.L. Nelson, D.R. Nelson, K.A. Nelson, K. Nixon, D.R. Nusskern J.M. Pacleb, M. Palazzolo, G.S. Pittman, S. Pan, J. Pollard, V. Puri, M.G. Reese, K. Reinert, K. Remington, R.D. Saunders, F. Scheeler, H. Shen, B.C. Shue, I. Siden-Kiamos, M. Simpson, M.P. Skupski, T. Smith, E. Spier, A.C. Spradling, M. Stapleton, R. Strong E. Sun, R. Svirskas, C. Tector, R. Turner, E. Venter, A.H. Wang, X. Wang, Z.Y. Wang, D.A. Wassarman, G.M. Weinstock, J. Weissenbach, S.M. Williams, T. Woodage, K.C. Worley, D. Wu, S. Yang, Q.A. Yao, J. Ye, R.F. Yeh, J.S Zaveri, M. Zhan, G. Zhang, Q. Zhao, L. Zheng, X.H. Zheng, F.N. Zhong, W. Zhong, X. Zhou, S. Zhu, X. Zhu, H.O. Smith, R.A. Gibbs, E.W. Myers, G.M. Rubin, and J.C. Venter. The genome sequence of *Drosophila melanogaster*. *Science*, 287:2185–2195, 2000.

4. T.L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proceedings of the 2nd International Conference on Intelligent Systems for Molecular Biology (ISMB)*, 1994, pp. 28–36.
5. L.O. Barrera and B. Ren. The transcriptional regulatory code of eukaryotic cells—insights from genome-wide analysis of chromatin organization and transcription factor binding. *Curr Opin Cell Biol*, 18:291–298, 2006.
6. G. Bejerano, M. Pheasant, I. Makunin, S. Stephen, W.J. Kent, J.S. Mattick, and D. Haussler. Ultraconserved elements in the human genome. *Science*, 304:1321–1325, 2004.
7. E. Berezikov, V. Guryev, R.H.A. Plasterk, and E. Cuppen. Conreal: Conserved regulatory elements anchored alignment algorithm for identification of transcription factor binding sites by phylogenetic footprinting. *Genome Res*, 14:170–178, 2004.
8. E.M. Blackwood and J.T. Kadonaga. Going to the distance: A current view of enhancer action. *Science*, 281:60–63, 1998.
9. M. Blanchette and M. Tompa. Discovery of regulatory elements by a computational method for phylogenetic footprinting. *Genome Res*, 12:739–748, 2002.
10. E. Blanco, D. Farre, M. Alba, X. Messeguer, and R. Guigó. ABS: A database of annotated regulatory binding sites from orthologous promoters. *Nucleic Acids Res*, 34:D63–D67, 2006.
11. E. Blanco and R. Guigó. Predictive methods using DNA sequences. In A.D. Baxevanis and B.F.F. Oullette, editors, *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins.* John Wiley, New York, 2005, pp. 115–142.
12. E. Blanco, X. Messeguer, T.F. Smith, and R. Guigó. Transcription factor map alignments of promoter regions. *PLoS Comput Biol*, 2:e49, 2006.
13. E. Blanco, M. Pignatelli, S. Beltran, A. Punset, S. Perez-Lluch, F. Serras, R. Guigó, and M. Corominas. Conserved chromosomal clustering of genes governed by chromatin regulators in *Drosophila*. *Genome Biol*, 9:R134, 2008.

14. A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *J Comput Biol*, 5:279–305, 1998.

15. J.C. Bryne, E. Valen, M.E. Tang, T. Marstrand, O. Winther, I. Piedade, A. Krogh, B. Lenhard, and A. Sandelin. Jaspar, the open access database of transcription factor-binding profiles: new content and tools in the 2008 update. *Nucleic Acids Res*, 36:D102–D106, 2008.

16. P. Bucher. Weight matrix descriptions of four eukaryotic RNA polymerase II promoter elements derived from 502 unrelated promoter sequences. *J Mol Biol*, 212:563–578, 1990.

17. The ENCODE Consortium. Identification and analysis of functional elements in 1% of the human genome by the encode pilot project. *Nature*, 447:799–816, 2007.

18. G.E. Crooks, G. Hon, J.M. Chandonia, and S.E. Brenner. Weblogo: A sequence logo generator. *Genome Res*, 14:1188–1190, 2004.

19. E.H. Davidson, J.P. Rast, P. Oliveri, A. Ransick, C. Calestani, C. Yuh, T. Minokawa, G. Amore, V. Hinman, C. Arenas-Mena, O. Otim, C.T. Brown, C.B. Livi, P.Y. Lee, R. Revilla, A.G. Rust, Z. Pan, M.J. Schilstra, P.J.C. Clarke, M.I. Arnone, L.Rowen, R.A. Cameron, D.R. McClay, L. Hood, and H. Bolouri. A genomic regulatory network for development. *Science*, 295:1669–1678, 2002.

20. E.T. Dermitzakis and A.G. Clark. Evolution of transcription factor binding sites in mammalian gene regulatory regions: Conservation and turnover. *Mol Biol Evol*, 7:1114–1121, 2002.

21. R. Durbin, S. Eddy, A. Crogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Protein and Nucleic Acids*, 1st edition. Cambridge University Press, 1998.

22. L. Duret and P. Bucher. Searching for regulatory elements in human noncoding sequences. *Curr Opin Struct Biol*, 7:399–406, 1997.

23. L. Elnitski, V.X. Jin, P.J. Farnham, and S.J. Jones. Locating mammalian transcription factor binding sites: a survey of computational and experimental techniques. *Genome Res*, 16:1455–1464, 2006.

24. P.G. Engstrom, S.J. Ho Sui, O. Drivenes, T.S. Becker, and B. Lenhard. Genomic regulatory blocks underlie extensive microsynteny conservation in insects. *Genome Res*, 17:1898–1908, 2007.

25. J.W. Fickett and A. Hatzigeorgiou. Eukaryotic promoter recognition. *Genome Res*, 7:861–878, 1997.

26. J.W. Fickett and W.W. Wasserman. Discovery and modeling of transcriptional regulatory regions. *Curr Opin Biotechnol*, 11:19–24, 2000.

27. D. Galas and A. Schmitz. DNAse footprinting: A simple method for the detection of protein-DNA binding specificity. *Nucleic Acids Res*, 5:3157–3170, 1978.

28. M.M. Garner and A. Revzin. A gel electrophoresis method for quantifying the binding of proteins to specific DNA regions: Application to components of the *Escherichia coli* lactose operon regulatory system. *Nucleic Acids Res*, 9:3047–3060, 1981.

29. M.B. Gerstein, C. Bruce, J.S. Rozowsky, D. Zheng, J. Du, J.O. Korbel, O. Emanuelsson, Z.D. Zhang, S. Weissman, and M. Snyder. What is a gene, post-ENCODE? history and updated definition. *Genome Res*, 17:669–681, 2007.

30. S.A. Goff, D. Ricke, T. Lan, G. Presting, R. Wang, M. Dunn, J. Glazebrook, A. Sessions, P. Oeller, H. Varma, D. Hadley, D. Hutchison, C. Martin, F. Katagiri, B.M. Lange, T. Moughamer, Y. Xia, P. Budworth, J. Zhong, T.J. Miguel, U. Paszkowski, S. Zhang,

M. Colbert, W.L. Sun, L. Chen, B. Cooper, S. Park, T.C. Wood, L. Mao, P. Quail, R. Wing, R. Dean, Y. Yu, A. Zharkikh, R. Shen, S. Sahasrabudhe, A. Thomas, R. Cannings, A. Gutin, D. Pruss, J. Reid, S. Tavtigian, J. Mitchell, G. Eldredge, T. Scholl, R.M. Miller, S. Bhatnagar, N. Adey, T. Rubano, N. Tusneem, R. Robinson, J. Feldhaus, T. Macalma, A. Oliphant, and S. Briggs. A draft sequence of the rice genome (*Oryza sativa L. ssp. japonica*). *Science*, 296:92–100, 2002.

31. A. Goffeau, B.G. Barrell, H. Bussey, R.W. Davis, B. Dujon, H. Feldmann, F.Galibert, J.D. Hoheisel, C. Jacq, M. Johnston, E.J. Louis, H.W. Mewes, Y. Murakamiand, P. Philippsen, H. Tettelin, and S.G. Oliver. Life with 6000 genes. *Science*, 274:546–567, 1996.
32. V. Gotea and I. Ovcharenko. Dire: Identifying distant regulatory elements of co-expressed genes. *Nucleic Acids Res*, 36:W133–W139, 2008.
33. O.L. Griffith, S.B. Montgomery, B. Bernier, B. Chu, K. Kasaian, S S. Aerts, S. Mahony, M.C. Sleumer, M. Bilenky, M. Haeussler, M. Griffith, S.M. Gallo, B. Giardine, B. Hooghe, P. Van Loo, E. Blanco, A. Ticoll, S. Lithwick, E. Portales-Casamar, I.J. Donaldson, G. Robertson, C. Wadelius, P. De Bleser, D. Vlieghe, M.S. Halfon, W.Wasserman, R. Hardison, C.M. Bergman, S.J.M. Jones, and The Open Regulatory Annotation Consortium. Oreganno: An open-access community-driven resource for regulatory annotation. *Nucleic Acids Res*, 36:D107–D113, 2008.
34. R. Guigó. DNA composition, codon usage and exon prediction. In M. Bishop, editor, *Genetic Databases.*, Academic Press, San Diego, CA, 1999, pp. 53–80.
35. O. Hallikas, K. Palin, N. Sinjushina, R. Rautiainen, J. Partanen, E. Ukkonen, and J. Taipale. Genome-wide prediction of mammalian enhancers based on analysis of transcription-factor binding affinity. *Cell*, 124:47–59, 2006.
36. C.T. Harbison, D.B. Gordon, T.I. Lee, N.J. Rinaldi, K.D. MacIsaac, T.W. Danford, N.M. Hannet, J. Tagne, D.B. Reynolds, J. YOO, E.G. Jennings, J. Zeitlinger, D.K. Pokholok, M. Kellis, P.A. Rolfe, K.T. Takusagawa, E.S. Lander, D.K. Gifford, E. Fraenkel, and R.A. Young. Transcriptional regulatory code of a eukaryotic genome. *Nature*, 431:99–104, 2004.
37. T.J.P. Hubbard, B.L. Aken, S. Ayling, B. Ballester, K. Beal, E. Bragin, S. Brent, Y. Chen, P. Clapham, L. Clarke, G. Coates, S. Fairley, S. Fitzgerald, J. Fernandez-Banet, L. Gordon, S. Graf, S. Haider, M. Hammond, R. Holland, K. Howe, A. Jenkinson, N. Johnson, A. Kahari, D. Keefe, S. Keenan, R. Kinsella, F. Kokocinski, E. Kulesha, D. Lawson, I. Longden, K. Megy, P. Meidl, B. Overduin, A. Parker, B. Pritchard, D. Rios, M. Schuster, G. Slater, D. Smedley, W. Spooner, G. Spudich, S. Trevanion, A. Vilella, J. Vogel, S. White, S. Wilder, A. Zadissa, E. Birney, F. Cunningham, V. Curwen, R. Durbin, X. M. Fernandez-Suarez, J. Herrero, A. Kasprzyk, G. Proctor, J. Smith, S. Searle, and P. Flicek. Ensembl 2009. *Nucleic Acids Res*, 37:D690–D697, 2009.
38. International Chicken Genome Sequencing Consortium, ICGSC. Sequence and comparative analysis of the chicken genome provide unique perspectives on vertebrate evolution. *Nature*, 432:695–716, 2004.
39. International Human Genome Sequencing Consortium, IHGSC. Finishing the euchromatic sequence of the human genome. *Nature*, 431:931–45, 2004.
40. International Mouse Genome Sequencing Consortium, IMGSC. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420:520–562, 2002.
41. J. Wang J. Yu, S. Hu, G.K. Wong, S. Li, B. Liu, Y. Deng, L. Dai, Y. Zhou, X. Zhang, M. Cao, J. Liu, J. Sun, J. Tang, Y. Chen, X. Huang, W. Lin, C. Ye, W. Tong, L. Cong,

J. Geng, Y. Han, L. Li, W. Li, G. Hu, X. Huang, W. Li, J. Li, Z. Liu, L. Li, J. Liu, Q. Qi, J. Liu, L. Li, T. Li, X. Wang, H. Lu, T. Wu, M. Zhu, P. Ni, H. Han, W. Dong, X. Ren, X. Feng X, P. Cui, X. Li, H. Wang, X. Xu, W. Zhai, Z. Xu, J. Zhang, S. He, J. Zhang, J. Xu, K. Zhang, X. Zheng, J. Dong, W. Zeng, L. Tao, J. Ye, J. Tan, X. Ren, X. Chen, J. He, D. Liu D, W. Tian, C. Tian, H. Xia, Q. Bao, G. Li, H. Gao, T. Cao, J. Wang, W. Zhao, P. Li, W. Chen, X. Wang, Y. Zhang, J. Hu, J. Wang, S. Liu, J. Yang, G. Zhang, Y. Xiong, Z. Li, L. Mao, C. Zhou, Z. Zhu, R. Chen, B. Hao, W. Zheng, S. Chen, W. Guo, G. Li, S. Liu, M. Tao, J. Wang, L. Zhu, L. Yuan, and H. Yang. A draft sequence of the rice genome (oryza sativa l. ssp. indica). *Science*, 296:79–92, 2002.

42. D.S. Johnson, A. Mortazavi, R.M. Myers, and B. Wold. Genome-wide mapping of in vivo protein-DNA interactions. *Science*, 316:1497–1502, 2007.
43. A.E. Kel, E. Goessling, I. Reuter, E. Cheremushkin, V. Kel-Margoulis, and E. Wingender. Match: a tool for searching transcription factor binding sites in DNA sequences. *Nucleic Acids Res*, 31:3576–3579, 2003.
44. J.T. Kim, T. Martinetz, and D. Polanti. Bioinformatic principles underlying the information content of transcription factor binding sites. *J Theor Biol*, 220:529–544, 2003.
45. T. Kouzarides. Chromatin modifications and their function. *Cell*, 128:693–705, 2007.
46. R.M. Kuhn, D. Karolchik, A.S. Zweig, T. Wang, K.E. Smith, K.R. Rosenbloom, B. Rhead, B.J. Raney, A. Pohl, M. Pheasant, L. Meyer, F. Hsu, A.S. Hinrichs, R.A. Harte, B. Giardine, P. Fujita, M. Diekhans, T. Dreszer, H. Clawson, G.P. Barber, D. Haussler, and W.J. Kent. The UCSC genome browser database: update 2009. *Nucleic Acids Res*, 37:D755–D761, 2009.
47. E.S. Lander and R.A. Weinberg. Genomics: Journey to the center of biology. *Science*, 287:1777–1782, 2000.
48. C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S Liu, A.F. Neuwald, and J.C. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.
49. C. Pal L.D. Hurst and M.J. Lercher. The evolutionary dynamics of eukaryotic gene order. *Nat Rev Genet*, 5:299–310, 2004.
50. B. Lenhard, A. Sandelin, L. Mendoza, P. Engstrom, N. Jareborg, and W.W. Wasserman. Identification of conserved regulatory elements by comparative genome analysis. *J Biol*, 2:13, 2003.
51. M. Levine and R. Tijan. Transcriptional regulation and animal diversity. *Nature*, 424:147–151, 2003.
52. G.G. Loots, I. Ovcharenko, L. Patcher, I. Dubchak, and E.M. Rubin. rVista for comparative sequence-based discovery of functional transcription factor binding sites. *Genome Res*, 12:832–839, 2002.
53. V. Matys, O.V. Kel-Margoulis, E. Fricke, I. Liebich, S. Land, A. Barre-Dirrie, I. Reuterand, D. Chekmenev, M. Krull, K. Hornischer, N. Voss, P. Stegmaier, B. Lewicki-Potapov, H. Saxel, A.E. Kel, and E. Wingender. TRANSFAC and its module TRANSCompel: Transcriptional gene regulation in eukaryotes. *Nucleic Acids Res*, 34:D108–D110, 2006.
54. C. Mayor, M. Brudno, J.R. Schwartz, A. Poliakov, E.M. Rubin, K.A. Frazer, L.S. Pachter, and I. Dubchak. VISTA: Visualizing global DNA sequence alignments of arbitrary length. *Bioinformatics*, 16:1046–1047, 2000.

55. M. Morey, S.K. Yee, T. Herman, A. Nern, E. Blanco, and S.L. Zipursky. Coordinate control of synaptic layer specificity and rhodopsins in photoreceptor neurons. *Nature*, 456:795–809, 2008.

56. T. Okumura, H. Makiguchi, Y. Makita, R. Yamashita, and K. Nakai. Melina II: A web tool for comparisons among several predictive algorithms to find potential motifs from promoter regions. *Nucleic Acids Res*, 35:W227–W231, 2007.

57. I. Ovcharenko, D. Boffelli, and G. Loots. eShadow: A tool for comparing closely related sequences. *Genome Res*, 14:1191–1198, 2004.

58. G. Parra, E. Blanco, and R. Guigó. Geneid in *Drosophila*. *Genome Res*, 10:511–515, 2000.

59. L.A. Pennacchio and E.M. Rubin. Genomic strategies to identify mammalian regulatory sequences. *Nat Rev Genet*, 2:100–109, 2001.

60. R.C. Perier, V. Praz, T. Junier, C. Bonnard, and P. Bucher. The eukaryotic promoter database (EPD). *Nucleic Acids Res*, 28:302–303, 2000.

61. E. Portales-Casamar, S. Kirov, J. Lim, S. Lithwick, M.I. Swanson, A. Ticoll, J. Snoddy, and W.W. Wasserman. PAZAR: A framework for collection and dissemination of cis-regulatory sequence annotation. *Genome Biol*, 8:R10, 2007.

62. K.D. Pruitt, T. Tatusova, and D.R. Maglott. NCBI reference sequences (refseq): A curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Res*, 35:D61–D65, 2007.

63. S. Rahmann, T. Muller, and M. Vingron. On the power of profiles for transcription factor binding site detection. *Stat Appl Genet Mol Biol*, 2:7, 2003.

64. B. Ren, F. Robert, J.J. Wyrick, O. Aparicio, E.G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Nannett, E. Kanin, T.L. Volkert, C.J. Wilson, S.R. Bell, and R.A. Young. Genome-wide location and function of DNA binding proteins. *Science*, 290:2306–2309, 2000.

65. J. Rohrer and M.E. Conley. Transcriptional regulatory elements within the first intron of Bruton's tyrosine kinase. *Blood*, 91:214–221, 1998.

66. T.D. Schneider and R.M. Stephens. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Res*, 18:6097–6100, 1990.

67. D.E. Schones, P. Sumazin, and M.Q. Zhang. Similarity of position frequency matrices for transcription factor binding sites. *Bioinformatics*, 21:307–313, 2005.

68. S. Schwartz, Z. Zhang, K.A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, and W. Miller. Pipmaker—a web server for aligning two genomic DNA sequences. *Genome Res*, 10:577–586, 2000.

69. Bovine Genome Sequencing and Analysis Consortium. The genome sequence of taurine cattle: a window to ruminant biology and evolution. *Science*, 324:522–528, 2009.

70. Chimpanzee Sequencing and Analysis Consortium. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437:69–87, 2005.

71. P.T. Spellman and G.M. Rubin. Evidence for large domains of similarly expressed genes in the *Drosophila* genome. *J Biol*, 1:5, 2002.

72. R. Staden. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Res*, 12:505–519, 1984.

73. G.D. Stormo. DNA binding sites: Representation and discovery. *Bioinformatics*, 16:16–23, 2000.

74. S.J. Ho Sui, D.L. Fulton, D.J. Arenillas, A.T. Kwon, and W.W. Wasserman. oPOSSUM: Integrated tools for analysis of regulatory motif over-representation. *Nucleic Acids Res*, 35:W245–W252, 2007.

75. D.A. Tagle, B.F. Koop, M. Goodman, J.L. Slightom, and D.L. Hess. Embryonic $\epsilon$ and $\gamma$ globin genes of a prosimian primate, nucleotide and amino acid sequences, developmental regulation and phylogenetic footprints. *J Mol Biol*, 203:439–455, 1988.

76. M. Tompa, N.Li, T.L. Bailey, G.M. Church, B. De Moor, E. Eskin, A.V. Favorov, M.C. Frith, Y. Fu, W.J. Kent, V.J. Makeev, A.A. Mironov, W.S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nat Biotechnol*, 23:137–144, 2005.

77. C. Tuerk and L. Gold. Systematic evolution of ligands by exponential enrichment: RNA ligands to bacteriophage t4 DNA polymerase. *Science*, 249:505–510, 1990.

78. J. Turatsinze, M. Thomas-Chollier, M. Defrance, and J. van Helden. Using RSAT to scan genome sequences for transcription factor binding sites and cis-regulatory modules. *Nat protocol*, 3:1578–1588, 2008.

79. A. Ureta-Vidal, L. Ettwiller, and E. Birney. Comparative genomics: Genome-wide analysis in metazoan eukaryotes. *Nat Rev Genet*, 4:251–262, 2003.

80. J.C. Venter, M.D. Adams, E.W. Myers, P.W. Li, R.J. Mural, G.G. Sutton, H.O. Smith, M. Yandell, C.A. Evans, R.A. Holt, J.D. Gocayne, P. Amanatides, R.M. Ballew, D.H. Huson, J.R. Wortman, Q. Zhang, C.D. Kodira, X.H. Zheng, L. Chen, M. Skupski, G. Subramanian, P.D. Thomas, J. Zhang, G.L. Gabor Miklos, C. Nelson, S. Broder, A.G. Clark, J. Nadeau, V.A. McKusick, N. Zinder, A.J. Levine, R.J. Roberts, M. Simon, C. Slayman, M. Hunkapiller, R. Bolanos, A. Delcher, I. Dew, D. Fasulo, M. Flanigan, L. Florea, A. Halpern, S. Hannenhalli, S. Kravitz, S. Levy, C. Mobarry, K. Reinert, K. Remington, J. Abu-Threideh, E. Beasley, K Biddick, V. Bonazzi, R. Brandon, M. Cargill, I. Chandramouliswaran, R. Charlab, K. Chaturvedi, Z. Deng, V. Di Francesco, P. Dunn K. Eilbeck, C. Evangelista, A.E. Gabrielian, W. Gan, W. Ge, F. Gong, Z. Gu, P. Guan, T.J. Heiman, M.E. Higgins, R.R. Ji, Z. Ke, K.A. Ketchum, Z. Lai, Y. Lei, Z. Li, J. Li, Y. Liang, X. Lin, F. Lu, G.V. Merkulov, N. Milshina, H.M. Moore, A.K. Naik, V.A. Narayan, B. Neelam, D. Nusskern, D.B. Rusch, S. Salzberg, W. Shao, B. Shue, J. Sun ad Z. Wang, A. Wang, X. Wang, J. Wang, M. Wei, R. Wides, C. Xiao, C. Yan, A. Yao, J. Ye, M. Zhan, W. Zhang, H. Zhang, Q. Zhao, L. Zheng, F. Zhong, W. Zhong, S. Zhu, S. Zhao, D. Gilbert, S. Baumhueter, G. Spier, C. Carter, A. Cravchik, T. Woodage, F. Ali, H. An, A. Awe, D. Baldwin, H. Baden, M. Barnstead, I. Barrow, K. Beeson, D. Busam, A. Carver, A. Center, M.L. Cheng, L. Curry, S. Danaher, L. Davenport, R. Desilets, S. Dietz, K. Dodson, L. Doup, S. Ferriera, N. Garg, A. Gluecksmann, B. Hart, J. Haynes, C. Haynes, C. Heiner, S. Hladun, D. Hostin, J. Houck, T. Howland, C. Ibegwam, J. Johnson, F. Kalush, L. Kline, S. Koduru, A. Love, F. Mann, D. May, S. McCawley, T. McIntosh, I. McMullen, M. Moy, L. Moy, B. Murphy, K. Nelson, C. Pfannkoch, E. Pratts, V. Puri, H. Qureshi, M. Reardon, R. Rodriguez, Y.H. Rogers, D. Romblad, B. Ruhfel, R. Scott, C. Sitter, M. Smallwood, E. Stewart, R. Strong, E. Suh, R. Thomas, N.N. Tint, S. Tse, C. Vech, G. Wang, J. Wetter, S. Williams, M. Williams, S. Windsor, E. Winn-Deen, K. Wolfe, J. Zaveri, K. Zaveri, J.F. Abril, R. Guigo, M.J. Campbell, K.V. Sjolander, B. Karlak, A. Kejariwal, H. Mi, B. Lazareva, T. Hatton, A. Narechania, K. Diemer, A. Muruganujan, N. Guo, S. Sato, V. Bafna, S. Istrail, R. Lippert, R. Schwartz, B. Walenz, S. Yooseph, D. Allen, A. Basu, J. Baxendale, L. Blick, M. Caminha, J. Carnes-Stine, P. Caulk, Y.H. Chiang, M. Coyne,

C. Dahlke, A. Mays, M. Dombroski, M. Donnelly, D. Ely, S. Esparham, C. Fosler, H. Gire, S. Glanowski, K. Glasser, A. Glodek, M. Gorokhov, K. Graham, B. Gropman, M. Harris, J. Heil, S. Henderson, J. Hoover, D. Jennings, C. Jordan, J. Jordan abd J. Kasha, L. Kagan, C. Kraft, A. Levitsky, M. Lewis, X. Liu, J. Lopez, D. Ma, W. Majoros, J. McDaniel, S. Murphy, M. Newman, T. Nguyen, N. Nguyen, M. Nodell, S. Pan, J. Peck, M. Peterson, W. Rowe, R. Sanders, J. Scott, M. Simpson, T. Smith, A. Sprague, T. Stockwell, R. Turner, E. Venter, M. Wang, M. Wen, D. Wu, M. Wu, A. Xia, A. Zandieh, and X. Zhu X. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

81. H. Wakaguri, R. Yamashita, Y. Suzuki, S. Sugano, and K. Nakai. DBTSS: Database of transcription start sites, progress report 2008. *Nucleic Acids Res*, 36:D97–D101, 2008.
82. W.W. Wasserman and A. Sandelin. Applied bioinformatics for the identification of regulatory elements. *Nat Rev Genet*, 5:276–287, 2004.
83. T. Werner. Identification and functional modelling of DNA sequence elements of transcription. *Brief Bioinformatics*, 1:372–380, 2000.
84. M.Q. Zhang. Computational prediction of eukaryotic protein-coding genes. *Nat Rev Genet*, 3:698–709, 2002.

# 20

# ALGORITHMIC ISSUES IN THE ANALYSIS OF CHIP-SEQ DATA

Federico Zambelli and Giulio Pavesi

## 20.1 INTRODUCTION

Researchers in biology and medicine nowadays have at their disposal enormous amounts of data and information, which provide an unprecedented opportunity to gain novel insights into the molecular basis of life and disease. The completion of several genome projects has given the (almost) complete DNA sequence of human and of several different organisms of interest, from viruses, to bacteria, to plants, to animals. This, in turn, has permitted the large-scale annotation of genes and their products, on the bricks of which life is built. Technologies like oligonucleotide microarrays, on the other hand, permit measuring the level of transcription of genes, that is, when and how much a given gene is activated according to developmental stage, cell cycle, external stimuli, disease, and so on. All in all, the emerging picture is that gene expression, that is, the series of steps in which a DNA region is transcribed into a RNA sequence, which in turn, is translated into a protein, is a process finely modulated at every stage by the cell. Thus, only when the regulation of this process also will be fully understood we will be able to obtain a complete picture of the mechanisms acting inside every living cell.

The first step of gene expression, the transcription of a DNA region into a complementary RNA sequence, is finely modulated and regulated by the activity of transcription factors (TFs), which are proteins (or protein complexes) that in turn are

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

encoded by the genome. TFs bind DNA in a sequence-specific manner in the neighborhood of the transcription start site of genes, or also in distal elements that are brought by the three-dimension (3-D) arrangement of DNA close to the gene region, with the overall effect of initiating the transcription process or, in some cases, of blocking it. The transcription of the gene thus can be started only when the right combination of TFs are bound to the DNA at the right time in its neighborhood [24]. To understand the complexity of this process suffices it to say that at least 10% of the about 22,000 human genes can be estimated to have this function, that is, regulate the transcription of other genes, yielding an exponential number of possible combinations and interactions.

Modern lab techniques allow for the large-scale identification of TF-DNA binding sites on the genome with experiments that were simply impossible to perform just a few years ago. A protein-DNA complex, like the genetic material comprising DNA and proteins that condense to form chromosomes in eukaryotic cells is called *chromatin*. Chromatin immunoprecipitation (ChIP) [8] is a technique that allows for the extraction from the cell nucleus of a specific protein-DNA chromatin complex, in our case, a given TF bound to the DNA. First, the TF is cross-linked, that is, fixed to the DNA. Then, a specific antibody that recognizes only the TF of interest is employed, and the antibody, bound to the TF, which in turn is bound to the DNA, permits the extraction and isolation of the chromatin complex. At this point, DNA is released from the TF by reverse cross linking, and researchers have at their disposal the DNA regions corresponding to the genomic locations of the sites that were bound in vivo, that is, inside living cells. The experiment is performed on thousands of cells at the same time to have a quantity of DNA suitable for further analysis and to have in the sample a good coverage of all regions bound by the TF.

The next phase logically is the identification of the DNA regions themselves and of their corresponding location in the genome, which in turn is made possible by the availability of the full genomic sequences. Also for this step, technology has witnessed dramatic improvements. From the identification of only predetermined candidate sites through polymerase chain reaction (PCR), the introduction of tiling arrays has permitted the analysis of the extracted DNA on a whole-genome scale (ChIP on Chip [44]) by using microarray probes designed to cover the sequence of a whole genome. The recent introduction of novel and efficient sequencing technologies collectively known as *next-generation sequencing* [35] has permitted moving this type of experiment one step further by providing at reasonable cost perhaps the simplest solution: to identify the DNA extracted by the cell, simply sequence the DNA itself (ChIP Sequencing, or ChIP-Seq [40, 34], Figure 20.1). In this way, all limitations derived from microarray technology can be overcome. These new sequencing systems can generate huge amounts of data. For example, the Illumina-Solexa system can generate more than 50 million sequences of length 3050 nt in a single run taking less than three days. Using a different technology, the ABI-SOLiD system can generate data at a similar rate. The Roche-454 system generates fewer but longer sequences [35].

Without delving into technical details, two of the three sequencing platforms just introduced are used nowadays for ChIP Seq: Illumina/Solexa and Solid which

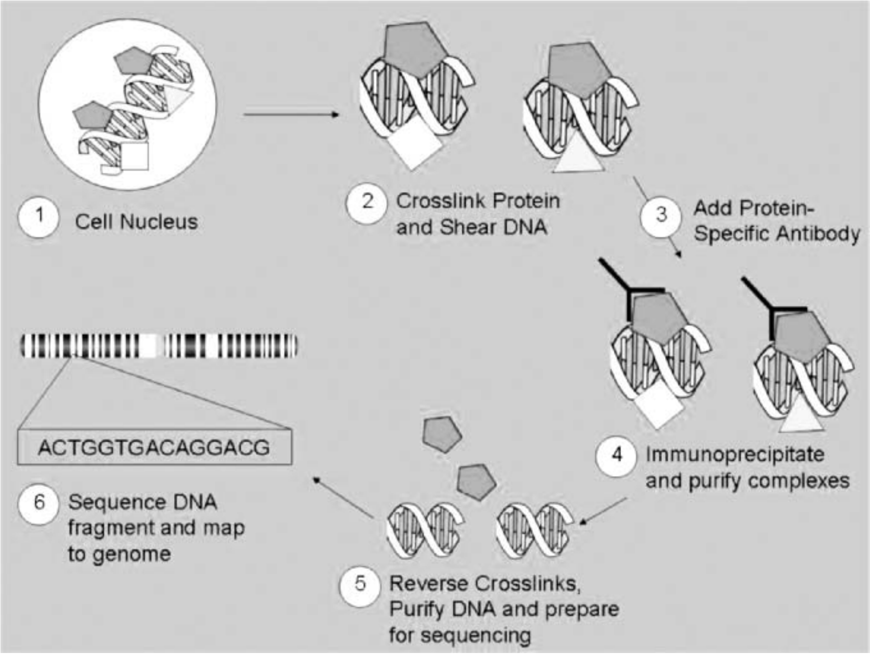

1 Cell Nucleus
2 Crosslink Protein and Shear DNA
3 Add Protein-Specific Antibody
4 Immunoprecipitate and purify complexes
5 Reverse Crosslinks, Purify DNA and prepare for sequencing
6 Sequence DNA fragment and map to genome
ACTGGTGACAGGACG

**Figure 20.1** The ChIP-Seq workflow. A protein (red), bound to the DNA inside the cell nucleus (1) is cross-linked (fixated) to the DNA, which is sheared into fragments (2). An antibody able to recognize only the red protein is added and binds to the red protein (3). The antibody-protein-DNA complex is isolated (4). DNA is separated from the red protein (reverse cross-linking, 5). Finally, the DNA obtained is sequenced (6).

produce on a given sample a larger number of sequences although shorter than 454, thus providing higher coverage of significant regions, as we will show later. Given a double-stranded DNA fragment derived as just described, the machine determines the nucleotide sequence at the beginning of either strand, moving from the 5′ to the 3′ direction (see Figure. 20.2). For technical limitations, the sequencer cannot determine the sequence of the complete region but only of a short fragment at its

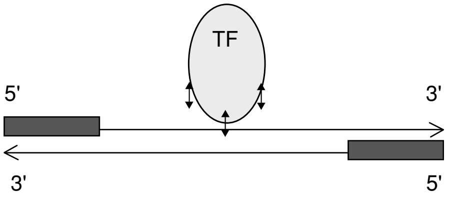


**Figure 20.2** Next-generation sequencing applied to a DNA region bound by a TF extracted by chromatin immunoprecipitation. Sequencing proceeds in the 5′ to 3′ direction, starting from the beginning of either strand and producing the sequence of either of the regions marked in red (usually, 25–50 base pairs). In case paired end sequencing is performed, the sequence of both the red regions is obtained simultaneously.

beginning, in other words, of a prefix usually ranging from 25 to 40 base pairs in ChIP-Seq applications. Thus, the output is a huge collection of millions of short sequences (called *reads* or *tags*), which mark the beginning of either strand of the DNA extracted from the cells. Recent developments also include the possibility of sequencing the beginning of both strands at the same time, producing paired end tags. As discussed, because before each region should have appeared more than once in the DNA sample and also because of the *amplification* of the DNA prior to sequencing, we should have tags marking the ends of the DNA regions bound by the TF appearing several times in the output. The overall number of sequence reads obtained varies from experiment to experiment and depends on several factors like the TF involved, sample preparation, experiment replicates, and so on. Suffice it to say that it usually ranges from a few to 10–20 million.

Once the sequencing has been completed, the computational analysis of the data can begin  [40]. The usual workflow is shown in Figure. 20.3. First, the tags are

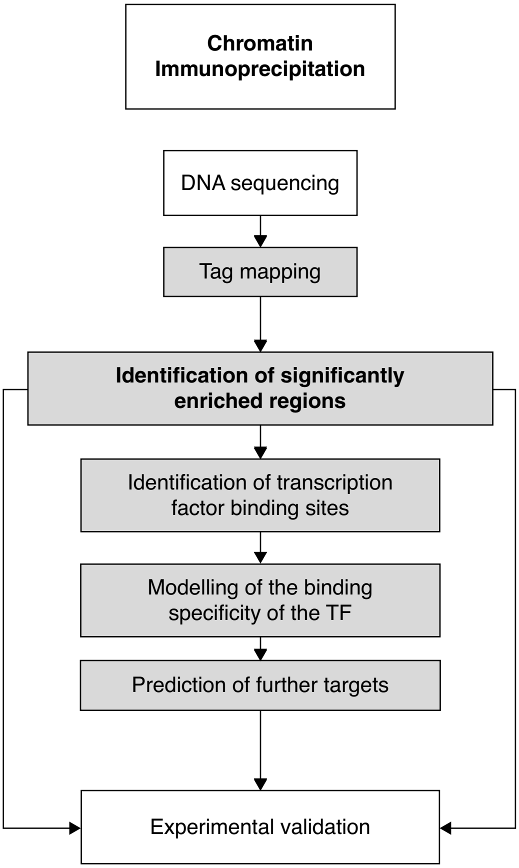

Chromatin Immunoprecipitation
DNA sequencing
Tag mapping
Identification of significantly enriched regions
Identification of transcription factor binding sites
Modelling of the binding specificity of the TF
Prediction of further targets
Experimental validation

**Figure 20.3** The ChIP-Seq analysis pipeline. Analysis steps that are performed in silico are highlighted in green.

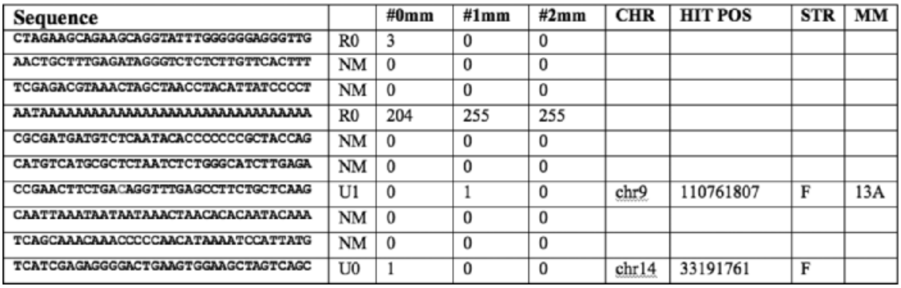
mapped on the genome to recover their original location. Then, regions bordered by tags on both ends (on opposite strands) in numbers high enough to represent a significant enrichment and not experimental noise are singled out (also, when available, with respect to a control experiment, aimed at producing some kind of random DNA and thus a random background model). These are the regions likely to have come from the sample because they are bound by the TF, and a genome-wide map of the binding of the TF is available. Then, further analyses are nevertheless possible; for example, because the regions extracted are longer than the actual binding sites for the TF, the latter have to be identified within the regions themselves; sites identified then can be employed to model the binding specificity of the TF to perform further in silico predictions in different cell types or experimental conditions; TF binding data can be crossed with expression data to understand better its activity in the regulation of target genes; co-operating or competing TFs can be identified in different ways, and so on.

## 20.2 MAPPING SEQUENCES ON THE GENOME

The first step of the analysis is the reassignment of the DNA fragments extracted on their original location on the genome, often called *tag mapping* or *read mapping*. In other words, each of the sequence tags produced has to be *matched* (or *aligned*) to the genome to recover its original position. Although, at least in theory, this would correspond to exact pattern matching, for which optimal and fast solutions already exist, in practice, we can expect only a limited number of tags to map exactly on the genome, mainly because of two reasons:

- Nucleotide polymorphisms: Although we have at our disposal a reference genomic sequence for several species of interest, in practice, the sequence of each individual of the same species is different, more notably at single positions in the genome (called *simple nucleotide polymorphisms*, or SNPs [6]). In other words, we can find that a tag cannot be matched exactly anywhere on a genome simply because the DNA sequence of the cells employed in the ChIP experiment differs from the sequence of the same region in the reference genomic sequence.
- Sequencing errors: A tag cannot be matched exactly anywhere on a genome because its sequence has been determined wrongly in one or more nucleotides.

Hence, we have to deal with a problem of *approximate pattern matching*, in which usually up to three substitutions are allowed on tags of 35–40 base pairs (bp). Also, one should consider the possibility of insertions and/or deletions in either the tags or the genome, for example, induced by the presence of stretches of the same nucleotide that may cause technical problems in the sequencing process. Moreover, given the nonuniform and repetitive nature of eukaryotic genomes, another factor that has to be considered is that despite the fact that about 30 bp should be enough to guarantee



| Sequence | | #0mm | #1mm | #2mm | CHR | HIT POS | STR | MM |
|---|---|---|---|---|---|---|---|---|
| CTAGAAGCAGAAGCAGGTATTTGGGGGGAGGGTTG | R0 | 3 | 0 | 0 | | | | |
| AACTGCTTTGAGATAGGGTCTCTCTTGTTCACTTT | NM | 0 | 0 | 0 | | | | |
| TCGAGACGTAAACTAGCTAACCTACATTATCCCCT | NM | 0 | 0 | 0 | | | | |
| AATAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA | R0 | 204 | 255 | 255 | | | | |
| CGCGATGATGTCTCAATACACCCCCCCGCTACCAG | NM | 0 | 0 | 0 | | | | |
| CATGTCATGCGCTCTAATCTCTGGGCATCTTGAGA | NM | 0 | 0 | 0 | | | | |
| CCGAACTTCTGACAGGTTTGAGCCTTCTGCTCAAG | U1 | 0 | 1 | 0 | chr9 | 110761807 | F | 13A |
| CAATTAAATAATAATAAACTAACACACAATACAAA | NM | 0 | 0 | 0 | | | | |
| TCAGCAAACAAACCCCCAACATAAAATCCATTATG | NM | 0 | 0 | 0 | | | | |
| TCATCGAGAGGGGACTGAAGTGGAAGCTAGTCAGC | U0 | 1 | 0 | 0 | chr14 | 33191761 | F | |

**Figure 20.4** A typical example of the result of tag mapping on the human genome, allowing, at most, two mismatches. Most tags do not match anywhere in the genome (NM in the second column from the left). The first and the fourth have repeated matches with no substitutions (R0), the first has three exact matches and the fourth has 204 exact, 255 with one mismatch, and 255 with two. Finally, we have two tags with a unique map, exact for the last one (U0, on chromosome 14, forward strand at hit pos coordinates), and one with one substitution (U1) at position 13 where a C on the tag (in red) matches an A on the genome.

a unique match on the genome, a single tag often can match different positions. A typical example is shown in Figure. 20.4; some tags do not map anywhere on the genome because of DNA contamination or sequencing artifacts; some map exactly or approximately on different positions; some map uniquely on a single position.

All in all, the problem can be formalized in different ways: from approximate pattern matching allowing only mismatches to approximate pattern matching also allowing indels. That is, find all substrings of the genome within a maximum Hamming distance or edit distance from the tag. Also, the output can range from the list of tags that map uniquely on the genome to the list of all matches, optimal or suboptimal, for each tag. Also, suitable heuristics can be and have been introduced to speed up the search. Often tags are accompanied by quality scores, which indicate for each nucleotide how reliable the read performed by the sequencer can be. Thus, for example, mismatches can be confined only to those tag nucleotides that are deemed to be less reliable by the sequencer itself. Or, because less reliable base calls often are located near the end of the sequence reads, one could require exact matching for the beginning of the tags (say, the first half) and allow for mismatches in the rest (say, the second half). Or, when the matching of some tags fails, the last bases can be trimmed away (probably wrong or a sequencing artifact), and the matching can be repeated for the shorter reads.

The main overall goal of research in this field is, intuitively, to obtain satisfactory solutions in as little time and space as possible to avoid the requirement of expensive dedicated hardware. As a result, many methods are based on the similar principles and algorithms but differ in the programming tricks or *ad hoc* heuristics that have been added to them, aimed at increasing their performance on real case studies at the price of a little decrease in their reliability. Research in this field is booming because next-generation sequencing is widely used in different applications well beyond ChIP-Seq from the sequencing or resequencing itself of genomes or transcriptomes, to genotyping, to expression profiling, and so on. As a consequence, new mapping software or improved versions of older tools today appear on an almost weekly basis and very often are released well before a publication fully describing their algorithmic details. Thus, here we will confine ourselves to a description of the general principles the most successful algorithms are based on, leaving to the interested reader the task of keeping abreast with one of the hottest and most rapidly growing fields of modern bioinformatics.

As mentioned before, a typical ChIP-Seq experiment on an organism like a human or a mouse can produce 5–20 million tags, and this number is increased further to hundreds of millions when, instead of selected regions of a genome, a whole genome or transcriptome has been sequenced. Thus, the sheer size of the data makes the problem virtually impossible to be tackled without the *indexing* of the sequences. And, as a matter of fact, all algorithms available for this step differ on whether they build an index of the genome or the tags and on the indexing method applied. Although the size of a genome like the human one is measured in billions of base pairs, on the other hand the overall size of the set of tags to be mapped in a ChIP-Seq experiment is in the millions or hundreds of millions. Thus, building an index for the genome and matching the tags against the index will have the consequence of requiring more memory space for the index but less time in the matching stage. Vice versa, indexing the tags and matching the genome against them has the consequence of reducing the memory requirements at the price of longer matching time. The indexing of the tags also has the benefit of being trivially scalable; in other words, if the available memory is not sufficient to hold the index of the whole set of tags to be processed, then the tags can be split into subsets, and each subset can be processed separately (or in parallel, if several computing cores are available). The final result is just given by merging the results obtained for each subset. Although clearly, the computation time is increased, on the other hand, tools of this kind are suitable for standard state-of-the-art personal computing equipment.

In general, approximate matching is based on the *pigeon-hole* principle. If we have 10 pigeons living in nine holes, then we can be sure that one hole has at least two pigeons living into it. Applied to our problem, suppose that we have to map a tag of length 32 with up to two mismatches. The tag can be defined as the concatenation of four strings of length eight: let A,B,C, and D be these four substrings. Then, at least two substrings should match the genome exactly, and we have that at least one of these strings matches the genome: AB, AXC, AXD, BC, BXD, and CD, where X denotes a stretch of eight wildcard or do not care characters. Thus, if we want to build an index for the genome, then we can index substrings of length 16, corresponding to the AB, AXC, and AXD combinations and use these as seeds for the initial exact matching stage. Alternatively, in the same situation, we can be certain that at least a substring of length 11 will match exactly. Hence, we can index substrings of length 11 and employ them as initial matching seed. In both cases, once a seed has been matched, it can be extended, allowing for mismatches and/or insertion and deletions.

The problem of mapping (or, in general, of aligning) sequences against a genome was introduced well before the problem we are dealing with appeared. It had and still has to be solved in any genome annotation project, for example for the mapping of RNAs or shorter *expressed sequence tags* (ESTs). *Blast-like alignment tool*

(BLAT) [20] was introduced exactly for this task and is based on the indexing of all nonoverlapping $k$-mers (substrings of length $k$) in a given genome. The index usually fits inside the random access memory (RAM) of inexpensive computers. Although very efficient, BLAT was designed for a different flavor of the problem (longer sequences to be mapped, and the mapping can include large gaps to accommodate for the intron-exon structure of the genes producing transcripts and/or accommodate for a larger number of errors to permit cross-species mapping). Its application to short tag mapping, which in turn implies using $k$-mers as short as eight base pairs and higher error frequency does not yield satisfactory results in terms of computation time and performance.

A 32 base pair tag-specific aligner called ELAND was developed in parallel with the Solexa sequencing technology, and it is provided for free for research groups that buy the sequencer. Probably the first tool introduced for this task, ELAND indexes the tags with the tag-splitting strategy we introduced before (and in its original version could map tags up to 32 nucleotides long), allowing only mismatches. It still is reported to be one of the fastest and less memory-greedy pieces of software available. Likewise, SeqMap [17] builds an index for the tags by using the longest substring guaranteed to match exactly, and scans the genome against it. It includes the possibility of introducing insertions and deletions for approximate matches.

ZOOM [29] again is based on the same principles as ELAND, with the difference being that tags are indexed by using spaced seeds that can be denoted with a binary string. For example, in the spaced seed 111010010100110111, 1s mean a match is required at that position, 0s indicate do not care positions. Only positions with a 1 in the seed are indexed. The performance reported is faster than ELAND, at the price of higher memory requirements. Short oligonucleotide alignment program (SOAP) [28] was one of the first methods published for the mapping of short tags in which both tags and genome first converted to numbers using 2-bits-per-base encoding. Each tag then is compared with exclusive-OR to the genome sequence. Then the value is used as a suffix to check a look-up table to know how many bases are different. To have a tradeoff between memory usage and efficiency, SOAP uses unsigned 3-bytes data type as the table element. To admit two mismatches, a read is split into fragments as in ELAND, and x mismatches can exist in at most x of the fragments at the same time. Mapping with either mismatches or indels is allowed but not simultaneously. Because for technical reasons, reads always exhibit a much higher number of sequencing errors at the 3’-end, which sometimes make them unalignable to the genome, SOAP iteratively can trim several base pairs at the 3’-end and redo the alignment until hits are detected or until the remaining sequence is too short for a specific match. All in all, the main drawback is the memory requirement, which is reported to be greater than 10 Gb for a genome like human.

PASS [7] likewise is based on a data structure that holds in RAM the hash table of the genomic positions of seed substrings (typically 11 and 12 bases) as well as an index of precomputed scores of short words (typically seven and eight bases) aligned against each other. After building the genomic index, the program matches each tag performing three steps: (1) it finds matching seed words in the genome; (2) for every match, it checks the precomputed alignment of the short flanking regions (thus including insertions and deletions); (3) if it passes step 2, then it performs an exact dynamic alignment of a narrow region around the initial match. The performance is reported to be much faster than SOAP but, once again, at the price of high memory requirements (10s of gigabytes (Gb)) for the genomic index.

Once again, the Maximum Oligonucleotide Mapping (MOM) [10] algorithm has two stages: first, searching for exactly matching short subsequences (seeds) between the genome and tag sequences and, second, performing ungapped (allowing only mismatches) extension on those seeds to find the longest possible matching sequence with an user-specified number of mismatches. To search for matching seeds, MOM creates a hash table of subsequences of fixed length $k$ ($k$-mers) from either the genome or the tag sequences and then sequentially reads the unindexed sequences searching for matching $k$-mers in the hash table. The addition in MOM is that, like in SOAP, tags that cannot be matched entirely given a maximum number of errors are trimmed automatically at both ends, assuming that sequencing errors or artifacts appear at the beginning or at the end of the tags (the original version of SOAP trimmed only the $3'$ end, *i.e.*, the suffixes). In this way, for each tag, it reports the maximal length match within the short read satisfying the error parameters. The performance reported is higher than SOAP in the number of tags successfully matched. Again, more than 10 Gb of memory are needed for typical applications.

The space requirements of building a genomic index with a hash table can be reduced by using more efficient strategies. A good (at least theoretically) performance also is obtained by vmatch [1], which employs enhanced suffix arrays for several different genome-wide sequence analysis applications. Bowtie [21] employs a Burrows–Wheeler index based on the full-text minute-space (FM) index, which has a reported memory requirement of only about 1.3 Gb for the human genome. In this way, Bowtie can run on a typical desktop computer with 2 Gb of RAM. The index is small enough to be precomputed and distributed together with the software. However, if one or more exact matches exist for a tag, then Bowtie always reports them, but if the best match is inexact, then Bowtie is not guaranteed in all cases to find it. Also the very recent Burrows–Wheeler aligner (BWA) [25] is based on the Burrows–Wheeler transform.

Mapping and Assembly with Quality (MAQ) [26] is one of the most successful tools in this field, especially devised to take advantage of the nucleotide-by-nucleotide quality scores that come together with the Solexa/Illumina tag sequences. The general idea is that mismatches caused by errors in sequencing mostly should appear at those positions in the tags that have a low quality score, thus are less reliable in the determination of the tag sequence. And, vice versa, those caused by single nucleotide polymorphisms (SNPs) always should appear at the same position in the genomic sequence. Mismatches thus are weighted according to their respective quality scores. For matching, MAQ builds multiple hash tables to index the tags and scans the genomic sequence. By default, six hash tables are used, ensuring that a sequence with two mismatches or fewer will be hit in an ELAND-like fashion. The six hash tables correspond to six spaced seeds as in ZOOM. Given 8-bp reads, for example, the six seeds would be 11110000, 00001111, 11000011, 00111100, 11001100, and 00110011, where nucleotides at 1’s are be indexed, whereas those at 0’s are not. By default, MAQ indexes the first 28 bp of the reads, which typically are the most accurate part. Very fast, MAQ on the other hand is based on several heuristics that do not always guarantee finding the best match for a tag.

Sequence quality scores provided by Illumina also are employed by RMAP [53]. A cutoff for the base-call quality score is used to designate positions as either high-quality or low-quality, depending on whether the quality score of the highest scoring base at that position exceeds the cutoff. Low-quality positions always induce a match (*i.e.*, act as wild cards). To prevent the possibility of trivial matches, a quality-control step eliminates reads with too many low-quality positions. CloudBurst [49] is a RMAP-like algorithm that supports cloud computation using the open-source Hadoop implementation of MapReduce (Google, Mountain View, CA) to parallelize execution using multiple nodes.

All in all, the clear picture is that the research in the field of large-scale short sequence mapping has boomed since 2007, when the first aligners were introduced. The performance of the different methods can be measured according to different parameters: time required, memory occupation, disk space, and in case of heuristic mappers, the actual number of tags that have been assigned correctly to their original position on the genome. In turn, the choice of a given method against another one depends on how many tags have to be mapped and naturally to the specifics of the computing equipment available. In our experience with ChIP-Seq experiments, in which the number of tags available is limited (a few million) as compared with a genome or transcriptome sequencing experiment (that can be as high as 100 times more), we usually chose an exhaustive algorithm able to guarantee, given a set of parameters, the optimal matching of every tag satisfying them (including trimming in case of unsuccessful mapping at the first try) also because, as we briefly discuss in the next section, the failure to map some tags can bias the results obtained from the experiment.

## 20.3 IDENTIFYING SIGNIFICANTLY ENRICHED REGIONS

Once the mapping has been completed, at least in theory, one already should have the final results of the experiment at his/her disposal. However, it is typical of a high-throughput experiment to produce a lot of noise, which is increased and amplified at each step, producing outputs like the one shown in Figure 20.4 where most tags do not map anywhere, and some match at multiple positions. Thus, it is hardly a surprise the fact that an experiment can be considered successful if about 20–25% of the tags produced can be mapped uniquely on the reference genome. The remaining majority of tags can be mapped on repetitive genomic regions or are simply experimental error resulting from DNA contamination, sequencing artifacts, errors in sequencing, and so on. And once the tags have been mapped on the genome, we are still far from the ideal situation of having them bordering the original DNA regions. What we usually have are tags that seem to map everywhere, with no clear separation (at least, for the human eye) of what are the regions derived from the original chromatin and which are “random” tags produced in the process by experimental noise, sequencing artifacts, or sample contamination. Thus, the next step in the analysis is far from being trivial and consists of determining the DNA regions that are bordered by several sequence tags sufficient to discriminate them from experimental noise.

A very efficient way to filter out noise is to conduct another experiment aimed at the production of noise itself; in other words, if random genomic DNA was added to the original experiment, then another experiment that produces only random DNA from the same type of cell (the control experiment) should give the opportunity to clean the result of the original one. The control experiment can be performed in different ways, by using an antibody not specific for any TF or, if possible, by using a cell in which the gene encoding the TF studied has been knocked out or silenced. An ideal output is the one shown in Figure 20.5. A true positive region should correspond to a genomic region that is bordered by several tags on both strands, and the tags on the two ends should be at a distance typical of experiments of this kind, that is,

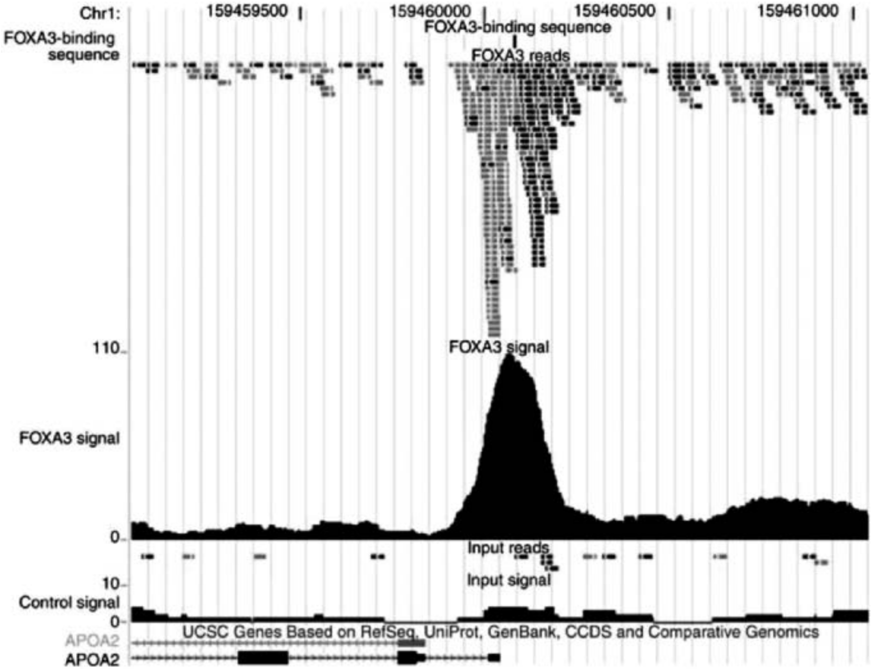


**Figure 20.5** An ideal peak in a ChIP-Seq experiment that identifies a binding site for TF FOXA3 shown within an UCSC genome browser window. FOXA3 reads are tags mapping on the sense (orange) and antisense (blue) DNA strand, bordering a region of about 250 base pairs corresponding to a FOXA3 binding site in the APOA2 promoter on human chromosome 1. Track FOXA3 signal depicts how many times each base pair in the region is covered by extending the tags in their respective direction, with a peak in the center of the region. Tracks “Input reads” and “Input signal” are the corresponding data for the control experiment. Notice how tags defining the peak in the center of the region are virtually absent from the control. Notice also how several tags map on the rightmost part of the region visualized but not in such a number or with an organization suitable to show a clear enrichment peak.

about 200–300 bps. By plotting the number of tags falling in each genomic position, the region thus should be bordered by two peaks, one made by tags on the positive strand and one on the negative. Analogously, if one extends each tag by the estimated length of the immunoprecipitated DNA fragments and plots how many times each nucleotide of the genome is covered by an extended tag, then a significantly enriched region should correspond to a single peak located in the middle of the region itself. On the other hand, the same region also should not appear at least with the same number of bordering tags or with the same height of the central peak in the control experiment.

In the absence of the control experiment, the problem to be solved is to estimate how many tags are sufficient to deem a region to be significantly enriched. Given $t$, the overall number of tags, and $g$, the size of the genome, if we assume that in a completely random experiment, each genomic region has the same probability of being extracted and sequenced, then the probability of finding one tag mapping in a given position is given by $t/g$. The same idea can be applied by dividing the genome into separate regions (*e.g.*, the chromosomes or chromosome arms) because for experimental reasons, different regions can have different propensities at producing tags. In this way, a global or region-specific local matching probability can be calculated and, therefore, the expected number of tags falling into any genomic region of size $w$. For this calculation, different statistical models can be employed, for example, Poisson, negative binomial, or gamma distributions [62]. Finally, the significance of tag enrichment is computed, by using sliding windows across the whole genome. If a control experiment is available, then the number of tags it produced from a given region can serve directly as background model.

Several peak-finding methods already have been published, including FindPeaks [12], F-Seq [5], SISSRS [19], QuEST [58], MACS [61], the ChipSeq Peak Finder used in [18], ChIPDiff [60], and CisGenome [16], which encompasses a series of tools for the different steps of the ChIP-Seq analysis pipeline. Regardless of the method used, false discovery rate estimates are calculated by these tools, based on the level of enrichment (number of tags) at the site, either globally or locally, compared with the background model used, for which in some cases a control experiment explicitly is required. Moreover, because so far, most experiments did not employ paired end tags as described in the introduction, some methods try and determine the original size of the DNA regions from the spacing of the tags on opposite strands that were extracted from the ChIP. In other words, they try to make ends meet guessing for each tag on a strand which could be the tag on the opposite strand marking the end of the original genomic region that was sequenced. This is usually done by determining which tag distances seem to be overrepresented in the sample. Some other methods, instead, simply leave this as a parameter to be set as input to the user.

The advantages and reliability of peak finding methods have yet to be fully appreciated because most of them initially were devised for a single experiment, and their portability to different organisms and/or experimental conditions is often not straightforward. Moreover, with current tools, the choice of a significance or enrichment threshold to discriminate real binding sites from background is often not clear and left to users based on calculated false discovery rates, on the level of enrichment of the expected binding motif, and/or on prior knowledge about genomic regions bound by the TF itself. That is, regions usually are ranked according to their enrichment, and it is up to the user to draw a line and choose thresholds above which regions can be considered to be significantly enriched. Threshold choice is especially difficult for TFs because most bind at low affinity to numerous sites, and a neat separation for the number of tags necessary to define enriched regions typically is not obtained. Moreover, some peaks could be missed because tags defining them map ambiguously on different positions on the genome, and if only best unique matches are kept, then the corresponding region does not contain enough tags to be deemed significant, or one of its ends disappears because it falls within a repetitive genomic region. Despite all these issues, several methods can be used to provide an argument that sites are functional. For example, the expression of genes with or without an associated site (thus regulated or not by the TF) can be compared. This can be convincing if the activity of the TF is straightforward (*i.e.,* the factor is a strong activator or a strong repressor). However, this is often not the case, and TFs can act as both activators and repressors within the same cell depending on the context of the site, or have varying levels of effect, depending on other TFs cooperating with them. All in all, the take-home message is that as always in bioinformatics but more importantly in case like these, reliable results cannot be produced without knowledge of the underlying biology; an over- or underestimation of the activity of a TF can produce biased views or, much worse, a completely wrong picture.

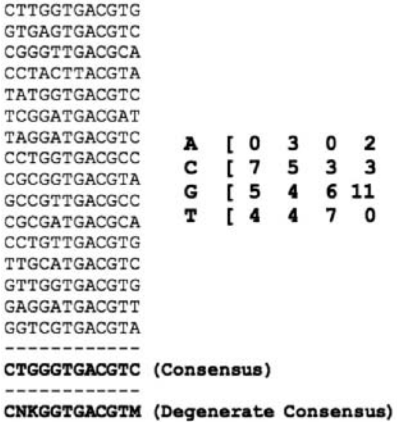
### 20.3.1 ChIP-Seq Approaches to the Identification of DNA Structure Modifications

So far, we have presented and discussed transcription regulation and ChIP-Seq from the viewpoint of transcription factors. However, the activation of transcription is regulated by several additional factors, the most important of which, called epigenetic regulation [15], depends on the structure of DNA itself. Inside cell nuclei, DNA is wrapped around specific proteins called *histones* forming structures called nucleosomes. Once again, DNA and histones together form chromatin. And, if the DNA region corresponding to a gene is wrapped up so to prevent TFs and the transcriptional mechanisms from contacting the DNA, then transcription simply does not start. DNA structure and accessibility inside the cell nuclei can be modified by the cell itself through posttranslational modifications of the histones like acetylation, methylation, ubiquitylation, phosphorylation, and sumoylation or by remodeling of chromatin. The result is that a previously unaccessible DNA region can become accessible or vice versa.

ChIP-Seq opened new avenues of research also for these aspects of gene regulation. In fact, ChIP can be used to extract from the cell histones that underwent some given modification together with the DNA attached to them. If the histone modification is associated with DNA unwrapping, then it permits the extraction of DNA accessible to TFs and the transcriptional machinery and to have a genome-wide map of active promoters/genes. As a matter of fact, the most successful applications of ChIP-Seq so far have been related to epigenetic regulation (see *e.g.*, [45]), perhaps because the problem is somewhat easier to be dealt with than with TFs. The analysis protocol is the same as we just described with TFs, with the difference in this case being that we do not have TFs binding DNA with different affinities resulting in grey areas of tag enrichment, as we discussed before. Rather, it is a yes or no decision, a histone can be modified or not, and thus, the separation between signal and noise in peak detection should be much clearer.

## 20.4 DERIVING ACTUAL TRANSCRIPTION FACTOR BINDING SITES

The actual DNA region bound by a TF (called *Transcription Factor Binding Site*, or TFBS) is much smaller than the DNA fragment isolated by ChIP. TFBSs in fact usually range in size from 8–10 to 16–20 nucleotides (small sequence elements of this size also are called *oligonucleotides*, or *oligos*) [54, 23]. A further step in the analysis thus is to recognize which are the actual TFBSs in the regions extracted to have an idea of the binding specificity of the TF and to build descriptors suitable for scanning genomic sequences for the identification of further sites in the absence of experimental data. As briefly mentioned in the introduction, TFs bind the DNA in a sequence-specific manner; but different from other DNA interacting proteins, like restriction enzymes, they bind DNA sequences that are similar but not identical; in other words, TF tolerate a certain degree of approximation in the DNA sequence they bind. This has the effect of complicating the problem in an awesome way. An example is shown in Figure 20.6. By comparing the different sites, one can notice that they differ in a few positions (mismatches) as nearly always the situation with



```
CTTGGTGACGTG
GTGAGTGACGTC
CGGGTTGACGCA
CCTACTTACGTA
TATGGTGACGTC
TCGGATGACGAT
TAGGATGACGTC    A  [ 0  3  0  2  5  0  0 16  0  0  1  5 ]
CCTGGTGACGCC    C  [ 7  5  3  3  1  0  0  0 16  0  5  6 ]
CGCGGTGACGTA    G  [ 5  4  6 11  7  0 15  0  0 16  0  3 ]
GCCGTTGACGCC    T  [ 4  4  7  0  3 16  1  0  0  0 10  2 ]
CGCGATGACGCA
CCTGTTGACGTG
TTGCATGACGTC
GTTGGTGACGTG
GAGGATGACGTT
GGTCGTGACGTA
------------
CTGGGTGACGTC (Consensus)
------------
CNKGGTGACGTM (Degenerate Consensus)
```

**Figure 20.6** Representing transcription factor binding site motifs. Given a set of sequences bound by the same TF, we can represent the motif they form by a consensus (bottom left) with the most frequent nucleotide in each position; a degenerate consensus, which accounts for ambiguous positions where there is no nucleotide clearly preferred (N = any nucleotide; K = G or T; M = A or C, according to IUPAC codes [39]); an alignment profile (right), which is converted into a frequency matrix by dividing each column by the number of sites used.

Note: box should cover the whole figure including matrix.

TFBSs. And we can notice how some positions are conserved strongly (*i.e.*, the TF does not seem to tolerate substitutions in those places, whereas in others any nucleotide seems to do). Given a set of validated instances, the binding preference of a given TF can be summarized and modeled. In the simplest form, we simply can take, position by position, the most frequent nucleotide and build a *consensus* of the sites. All oligos that differ from the consensus up to a maximum number of allowed substitutions can be considered valid instances of binding sites for the same TF. Clearly, this is an oversimplification that does not take into account the different level of variability at different position of the sites. A more involved method is to use degenerate consensi, which can be formalized, for example, by using regular expressions. Positions where there does not seem to be any preference for any nucleotide are left undefined, and any nucleotide can match these; positions where two or three nucleotides can be found with approximately the same frequency are left ambiguous, and any of the ambiguous nucleotides are considered a valid match; a single nucleotide is used only for the most conserved positions, which require a single-nucleotide match. Finally, the most flexible and widely used way of building descriptors for TF is to align the available sites and to build an alignment *profile* representing the frequency with which each nucleotide appears at each position in the sites. Thus, any candidate oligo can be aligned against the profile, and the corresponding frequencies can be used to evaluate how well it fits the descriptor (rather than a simple yes/no decision like with consensi) [54].

All in all, the problem can be defined informally as follows: given a set of DNA sequences (the regions identified by the ChIP-Seq), find a set of oligos appearing in all or most of the sequences (thus allowing for experimental errors and the presence of false positives in the set) similar enough to one another to be instances of sites recognized by the same TF. And, the same set of similar oligos should not appear with the same frequency and/or the same degree of similarity in a set of sequences selected at random or built at random with a generator of biologically feasible DNA sequences. This set of similar and overrepresented oligos collectively build a *motif* recurring in the input sequences.

This problem, generally described in literature as *motif discovery*, appeared well before the introduction of large-scale techniques like Chip-Seq [42, 48]. The same principles, in fact, apply to the analysis of gene promoter sequences; a set of genes showing similar expression patterns should be coregulated to some extent by the same TFs; because the latter usually bind DNA in the promoter region, by analyzing promoter sequences from coexpressed genes, we should be able to identify as overrepresented the sites recognized by the common regulators. Unfortunately, this type of problem has proven itself to be one of the hardest in bioinformatics history for several reasons [57]. When instead of a set of promoters the input is a set of sequences derived from a ChIP experiment, the problem seems to become easier, for different reasons:

- The size of the sequence sets: A ChIP produces usually thousands of candidate regions, whereas a typical promoter analysis of coexpressed genes is performed on a few dozen sequences.

- The length of the sequences: Sequences extracted from ChIP-Seq are usually 200–300 bp long, as opposed to promoters that usually are defined as 500–1000 bp long.
- The frequency with which binding sites for the same TF appears: In a ChIP-Seq, they should appear in a very high percentage of the sequences examined, whereas in a set of coexpressed genes, there is no guarantee for this. In the latter case, in fact, a TF could regulate only a limited subset of the genes, and the common pattern of expression observed could be a result of the simultaneous action of several TFs.

All in all, what happens is that with ChIP-Seq, we have a somewhat cleaner input sequence set (most sequences should contain what we are looking for), and more redundant, because in thousands of sequences, we can expect to find several instances of binding sites very similar to one another. In gene promoter analysis, the input set is much less cleaner (there is no guarantee on how many sequences actually share the same motif), the sequence set is much smaller (and thus, the different sites in the sequences can be very different from one another), and the sequences are longer. Thus, at least in theory, in ChIP-Seq, we can expect to obtain a clearer separation of the signal (the motif) from random similarities.

Approaches to the problem mainly differ in these three points:

1. In how similar oligos forming a candidate motif are detected and in how the motif they form is modeled
2. In the random background model used to assess the statistical significance of the motifs
3. In the optimization strategy used to find the most significant motif

Given $k$ input sequences of length $n$ and a motif size $m$, by assuming that a motif instance should appear in each sequence, we have $(n - m + 1)^k$ candidate solutions, an exponential number of input sequences that, regardless of the significance or scoring function used to evaluate the solutions, leads to a nondeterministic polynomial (NP)-hard problem. Leaving aside the design of performance-guaranteed approximation algorithms [2], which produce solutions too far from the optimal one to be biologically meaningful, different heuristics can be applied to the problem. And, as a matter of fact, research on this field already has produced many different approaches and methods. Here, for the sake of space, we will describe the general ideas in tackling the problem in its more general definitions, which are the basis of tools most widely used now in the analysis of ChIP-Seq data. First of all, one has to decide how to model a solution. The most widely known algorithms can be split roughly into *consensus-driven* and *sequence-driven* methods [42]. In the former, each set of oligos is summarized by using a consensus, and all oligos differing from the consensus up to a maximum number of mismatches can be considered *a priori* a valid motif instance. Thus, the problem can be formalized as follows: for each of the $4^m$ DNA strings of reasonable size $m$ (8–16 nucleotides), collect from the input sequences all the approximate occurrences up to $e$ mismatches, where $e$ depends on the motif size $m$. In other words, the problem becomes exhaustive approximate pattern matching. Introduced in the early days of bioinformatics [13, 47, 59], this approach had been abandoned because it is considered too time-consuming, because it requires enumerating an exponential (in the solution length) number of candidate solutions. The application of indexing structures like suffix trees to the input sequence set nevertheless showed its feasibility, reducing its complexity to be exponential in the number of mismatches allowed [41, 36]. The search space can be trimmed further by employing degenerate consensi to model the solutions and to tolerate mismatches only in the degenerate positions, with a significant improvement in the time needed for matching [50, 52]. Consensi can be ranked finally according to the number of matches/degree of conservation of the matches. Conversely, the problem can be tackled by a combinatorial approach. Assuming, for example, that the TF should have one site per input sequence, the problem becomes choosing from each sequence an oligo so that the resulting motif yields the maximum score given a measure of significance to be optimized. Exhaustive enumeration of all combinations is unfeasible; thus, approaches to this problem differ in the heuristics used to build initial solutions and in the exploration of the search space. Here we briefly will describe the most successful methods in the "one occurrence per sequence" mode, but each one can be adapted to the discovery of motifs appearing only in a subset of the input sequences.

Consensus [14], one of the earliest methods, is based on a greedy algorithm. Sequences are processed one-by-one. The best solutions for two sequences are kept and augmented by adding one oligo from the third sequence. Again, the best solutions are kept and augmented with oligos from the fourth sequence, and so on, until all sequences have been processed, and the remaining solutions contain one oligo per input sequence. MEME [4] is an EM algorithm that is equivalent to performing a local search; given an initial solution built with one oligo per input sequence, oligos that fit the motif better than the existing ones are chosen and replace them building a new solution. For time reasons, this step is not iterated until convergence but is performed a limited number of times on numerous initial candidate solutions. Only the highest scoring ones are finally further optimized until convergence. Several methods are based, with little differences, on the Gibbs sampling strategy first introduced in [22]. The general idea is a Monte Carlo variation on the local search; instead of choosing the oligo for replacement that best fits the current motif, each oligo in the input sequences has a probability of being chosen to form a motif dependent on its fitness with respect to the motif currently being optimized. Thus, suboptimal optimization steps can be taken. The algorithm is usually very fast, obtaining rapid convergence in a limited number of steps, resulting in its application in different tools for the optimization of different significance functions [31, 33, 37, 38, 51, 55, 56]. Nested-MICA [9] employs a different optimization strategy, called Nested Sampling, which claims to be "likely to find a globally optimal model (motif) in a single run," and uses a model based on the independent component analysis (ICA) framework to learn motifs.

Although at the beginning, all methods mostly were tested and validated on synthetic datasets, more recent assessments based on real case studies have been introduced, some derived from ChIP experiments. The results have shown that the real issue in this field seems not to be the strategy used to optimize a significance function but rather on the choice itself of a suitable measure of significance able to discriminate real motifs from random similarities. As mentioned before, any measure should take into account at least two factors: how much of the motif is conserved (how much the oligos forming it are similar to one another) and how unlikely the same motif is to appear in a set of sequences chosen at random or built at random with some sequence generator.

If a motif is modeled with a profile, then we can view the sequences as generated by two different sources: the motif itself, which produces nucleotides with the probability equivalent to the frequencies of the nucleotides in each column of the profile, and the background, which produces nucleotides according to some background distribution. If we assume that nucleotides in the input sequences are independent, that is, the probability of finding a nucleotide in any position is not influenced by its neighbors, then the overall conservation (similarity among the oligos) of the motif and its distance from the background distribution thus can be measured by computing the *information content* (or *relative entropy*) of the profile:

$$\mathrm{IC} = \sum_{i=1}^{4} \sum_{j=1}^{m} m_{i,j} \log \frac{m_{i,j}}{b_i}$$

where $m_{i,j}$ is the entry in row $i$ and column $j$ of the profile (ranging from 0 to 1) and $b_i$ is the expected frequency of nucleotide $i$ in the input sequences (which in turn can be estimated by using the genomic sequence of the organism studied or by the input sequences themselves). Clearly, for each column $j$, we have that $\sum_{i=1}^{4} m_{i,j} = 1$ and $\sum_{i=1}^{4} b_i = 1$. It clearly is shown how this measure accounts for how much each column is conserved and how much the nucleotide frequencies obtained in the profile differ from what would have been obtained by aligning oligos chosen at random. This was, for example, the measure optimized in the first versions of Consensus, MEME, and the Gibbs Sampler. For assessing the significance of motifs described with consensi, the principle is the same; the probability of finding any oligo in the input sequences can be computed as the product of the background probabilities for each nucleotide of the oligo, and analogously, the probability of finding an approximate occurrence of the consensus with $e$ substitutions is the sum of the probabilities associated with the oligos within Hamming distance $e$ from the consensus. Alternatively, the match count for a consensus in the input sequences can be compared with the match count obtained in shuffled input sequences (*i.e.*, consists of the same nucleotides in a different order [36, 46]), which serves as expected count. Then, enrichment in the input sequences can be computed with different statistical models based on different background probability distributions.

In both approaches we just described, the weakest link is the independent assumption. If we take any real DNA sequence, then we can observe that in most cases, the expected number of times a given oligo appears in the sequence is significantly higher or lower than the expected count that would be derived from the product of nucleotide probabilities. Thus, significant improvements have been obtained by assuming that the probability of finding a nucleotide at any given position is influenced by neighboring nucleotides as well. This can be done by modeling the background with higher order Markov models. Intuitively, in a $j$th order Markov model, the probability of finding a nucleotide in a given position of an oligo depends on the $j$ nucleotides preceding it (the independent model is thus a 0 order Markov model). These parameters can be estimated from available sequences; for example, a motif finding tool for human promoter sequences can use a Markov model built from the analysis of all available human promoter sequences. By augmenting the information content score with a background rareness score based on the probability of finding at random the oligos forming the motif computed with a 5th or 6th order Markov model, the performance of profile-based methods was improved significantly (see *e.g.*, [33]). In the consensus-based method Weeder [43], the probabilities associated with oligos up to length eight are computed directly from the oligo counts in all promoters of a given genome and for longer oligos with a Markov model of the 7th order. Indeed, research also has focused on finding the most suitable Markov model order for motif discovery [27]. NestedMICA [9] introduces mosaic background modeling. The idea is to use four different higher order background models according to the overall nucleotide composition of the input sequences, particularly for the content of C and G nucleotides (corresponding to the presence or absence of CpG islands), reporting good performance results.

As mentioned before, solutions to the problem applied to sequences derived from ChIP-Seq and similar large-scale experiments benefit from the fact that the motif is more redundant in the input set, both in number of occurrences and in the degree of conservation. And, on the other hand, the large number of sequences to be processed make combinatorial approaches demanding for computational resources. Perhaps with these considerations in mind, algorithms that were devised for large-scale ChIP experiments like MDScan [32] and the latest additions to the field (Trawler [11] and Amadeus [30]) show superior performance on ChIP-Seq and similar large sequence sets in terms of motifs correctly identified but much more strikingly in computational resources required. The general ideas underlying these three methods are somewhat similar. Initial candidate solutions are built by matching consensi (MDScan) or degenerate consensi on the input sequences indexed with a suffix tree in Trawler to speed up the search. Although less flexible than other consensus-based approaches like Weeder, degenerate consensi anyway can capture significant motifs given the high number of motif instances in the sequences. Significance then is assessed with a third-order background model (MDScan), or more simply by comparing the counts in the input to randomly selected background sequence sets, with $z$-scores (Trawler) or a hypergeometric test (Amadeus). Finally, similar solutions are merged into more complex oligo sets, which are modeled using a profile that is eventually further optimized on the input sequences with MEME-like strategies.

Research on this topic also should take advantage of additional features that can be associated with regions derived from ChIP-Seq. For example, because the number of tags defining a region is reported to be an indicator of the affinity of the TF for the region [19], higher priority or weight should be given to those regions that are more enriched in tags, as in MDScan for regions extracted by ChIP on Chip according to microarray probe enrichment. This is something that can be done trivially by applying existing methods to only a subset of the input sequences, but this factor could be taken into account directly by the algorithms, and also the modeling of the binding specificity of the TF with a profile (resulting from motif discovery algorithms) should given a higher weight to high-affinity sites corresponding to the most enriched regions. As a matter of fact, for some TFs significant improvements in the reliability of descriptors seem to be attained by using different descriptors for high and low-affinity sites [3]. Finally, the sites bound by the TF are more likely to be located in the center of the region extracted by ChIP [19]: thus, adding positional bias to sequence conservation in assessing the motifs somewhat should improve the results.

## 20.5 CONCLUSIONS

Next-generation sequencing techniques have opened new avenues for research in molecular biology and at the same time highlighted once more the key role that bioinformatics play in modern genomics. In fact, apart from the initial sequencing and the eventual validation of some predicted targets, virtually all steps of a ChIP-Seq analysis are performed *in silico*. In this chapter, we gave an overview of the problems deriving from the different phases of the pipeline, and without claiming to be exhaustive, presented the ideas underlying the most widely used methods. If we consider that next-generation sequencing is about four years old, and the first pioneering articles on ChIP-Seq appeared just a couple of years ago, the large number of works in this field that appeared in such a short time (including those that were published in the weeks in which we were writing this chapter) is a clear proof of the great interest this and similar fields of research have raised. Until the appearance of third-generation sequencing or completely new amazing lab techniques, of course.

## REFERENCES

1. M.I. Abouelhoda, S. Kurtz, and E. Ohlebusch. The enhanced suffix array and its applications to genome analysis. *Proceedings of Algorithms in Bioinformatics*, volume 2452, 2002, pp. 449–463.
2. T. Akutsu, H. Arimura, and S. Shimozono. On approximation algorithms for local multiple alignment. *Proceedings of RECOMB 2000* ACM, 2000, pp. 1–12.
3. G. Badis, M.F. Berger, A.A. Philippakis, S. Talukder, A.R. Gehrke, S.A. Jaeger, E.T. Chan, G. Metzler, A. Vedenko, X. Chen, H. Kuznetsov, C.F. Wang, D. Coburn, D.E. Newburger, Q. Morris, T.R. Hughes, and M.L. Bulyk. Diversity and complexity in DNA recognition by transcription factors. *Science*, 324:1720–1723, 2009.

4. T.L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *International Conference of Intelligent Systems for Molecular Biology*, volume 2, 1994, pp. 28–36.
5. A.P. Boyle, J. Guinney, G.E. Crawford, and T.S. Furey. F-seq: A feature density estimator for high-throughput sequence tags. *Bioinformatics*, 24(21):2537–2538, 2008.
6. A.J. Brookes. The essence of snps. *Gene*, 234(2):177–186, 1999.
7. D. Campagna, A. Albiero, A. Bilardi, E. Caniato, C. Forcato, S. Manavski, N. Vitulo, and G. Valle. Pass: A program to align short sequences. *Bioinformatics*, 25(7):967–968, 2009.
8. P. Collas and J.A. Dahl. Chop it, ChIP it, check it: The current status of chromatin immunoprecipitation. *Front Biosci*, 13:929–943, 2008.
9. T.A. Down and T.J. Hubbard. NestedMICA: Sensitive inference of over-represented motifs in nucleic acid sequence. *Nucleic Acids Res*, 33(5):1445–1453, 2005.
10. H.L. Eaves and Y. Gao. Mom: Maximum oligonucleotide mapping. *Bioinformatics*, 25(7):969–970, 2009.
11. L. Ettwiller, B. Paten, M. Ramialison, E. Birney, and J. Wittbrodt. Trawler: De novo regulatory motif discovery pipeline for chromatin immunoprecipitation. *Nat Methods*, 4(7):563–565, 2007.
12. A.P. Fejes, G. Robertson, M. Bilenky, R. Varhol, M. Bainbridge, and S.J. Jones. Findpeaks 3.1: A tool for identifying areas of enrichment from massively parallel short-read sequencing technology. *Bioinformatics*, 24(15):1729–1730, 2008.
13. D.J. Galas, M. Eggert, and M.S. Waterman. Rigorous pattern-recognition methods for DNA sequences. Analysis of promoter sequences from escherichia coli. *J Mol Biol*, 186(1):117–128, 1985.
14. G.Z. Hertz and G.D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15(7-8):563–577, 1999.
15. R. Jaenisch and A. Bird. Epigenetic regulation of gene expression: How the genome integrates intrinsic and environmental signals. *Nat Genet*, 33 (Suppl):245–254, 2003.
16. H. Ji, H. Jiang, W. Ma, D.S. Johnson, R.M. Myers, and W.H. Wong. An integrated software system for analyzing ChIP-chip and ChIP-seq data. *Nat Biotechnol*, 26(11):1293–1300, 2008.
17. H. Jiang and W.H. Wong. Seqmap: Mapping Massive amount of oligonucleotides to the genome. *Bioinformatics*, 24(20):2395–2396, 2008.
18. D.S. Johnson, A. Mortazavi, R.M. Myers, and B. Wold. Genome-wide mapping of in vivo protein-DNA interactions. *Science*, 316(5830):1497–1502, 2007.
19. R. Jothi, S. Cuddapah, A. Barski, K. Cui, and K. Zhao. Genome-wide identification of in vivo protein-DNA binding sites from ChIP-seq data. *Nucleic Acids Res*, 36(16):5221–5231, 2008.
20. W.J. Kent. BLAT–the BLAST-like alignment tool. *Genome Res*, 12(4):656–664, 2002.
21. B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.
22. C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.
23. B. Lemon and R. Tjian. Orchestrated response: A symphony of transcription factors for gene control. *Genes Dev*, 14(20):2551–2569, 2000.

24. M. Levine and R. Tjian. Transcription regulation and animal diversity. *Nature*, 424(6945):147–151, 2003.
25. H. Li and R. Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25:1754–1760, 2009.
26. H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–1858, 2008.
27. N. Li and M. Tompa. Analysis of computational approaches for motif discovery. *Algorithm Mol Biol*, 1(1):8, 2006.
28. R. Li, Y. Li, K. Kristiansen, and J. Wang. SOAP: Short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.
29. H. Lin, Z. Zhang, M.Q. Zhang, B. Ma, and M. Li. ZOOM! zillions of oligos mapped. *Bioinformatics*, 24(21):2431–2437, 2008.
30. C. Linhart, Y. Halperin, and R. Shamir. Transcription factor and microRNA motif discovery: The amadeus platform and a compendium of metazoan target sets. *Genome Res*, 18(7):1180–1189, 2008.
31. X. Liu, D.L. Brutlag, and J.S. Liu. Bioprospector: Discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pacific Symposium on Biocomputing*, 2001, pp. 127–138.
32. X.S. Liu, D.L. Brutlag, and J.S. Liu. An algorithm for finding protein-DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nat Biotechnol*, 20(8):835–839, 2002.
33. K. Marchal, G. Thijs, S. De Keersmaecker, P. Monsieurs, B. De Moor, and J. Vanderleyden. Genome-specific higher-order background models to improve motif detection. *Trends Microbiol*, 11(2):61–66, 2003.
34. E.R. Mardis. ChIP-seq: Welcome to the new frontier. *Nat Methods*, 4(8):613–614, 2007.
35. E.R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends Genet*, 24(3):133–141, 2008.
36. L. Marsan and M.F. Sagot. Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. *J Comput Biol*, 7(3-4):345–362, 2000.
37. C. Narasimhan, P. LoCascio, and E. Uberbacher. Background rareness-based iterative multiple sequence alignment algorithm for regulatory element detection. *Bioinformatics*, 19(15):1952–1963, 2003.
38. A.F. Neuwald, J.S. Liu, and C.E. Lawrence. Gibbs motif sampling: Detection of bacterial outer membrane protein repeats. *Protein Sci*, 4(8):1618–1632, 1995.
39. Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Nomenclature for incompletely specified bases in nucleic acid sequences. Recommendations 1984. *Proc Natl Acad Sci U S A*, 83(1):4–8, 1986.
40. P.J. Park. ChIP-seq: Advantages and challenges of a maturing technology. *Nat Rev Genet*, 10:669–680, 2009.
41. G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in DNA sequences. *Bioinformatics*, 17(Suppl 1):S207–S214, 2001.
42. G. Pavesi, G. Mauri, and G. Pesole. In silico representation and discovery of transcription factor binding sites. *Brief Bioinformatics*, 5(3):217–236, 2004.

43. G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole. Weeder web: Discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Res*, 32(Web Server issue):W199–W203, 2004.

44. S. Pillai and S.P. Chellappan. ChIP on chip assays: Genome-wide analysis of transcription factor binding and histone modifications. *Methods Mol Biol*, 523:341–366, 2009.

45. A.G. Robertson, M. Bilenky, A. Tam, Y. Zhao, T. Zeng, N. Thiessen, T. Cezard, A.P. Fejes, E.D. Wederell, R. Cullum, G. Euskirchen, M. Krzywinski, I. Birol, M. Snyder, P.A. Hoodless, M. Hirst, M.A. Marra, and S.J. Jones. Genome-wide relationship between histone h3 lysine 4 mono- and tri-methylation and transcription factor binding. *Genome Res*, 18(12):1906–1917, 2008.

46. S. Robin, J.J. Daudin, H. Richard, M.F. Sagot, and S. Schbath. Occurrence probability of structured motifs in random sequences. *J Comput Biol*, 9(6):761–773, 2002.

47. J.R. Sadler, M.S. Waterman, and T.F. Smith. Regulatory pattern identification in nucleic acid sequences. *Nucleic Acids Res*, 11(7):2221–2231, 1983.

48. G.K. Sandve and F. Drablos. A survey of motif discovery methods in an integrated framework. *Biol Direct*, 1:11, 2006.

49. M.C. Schatz. Cloudburst: Highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

50. D. Shinozaki, T. Akutsu, and O. Maruyama. Finding optimal degenerate patterns in DNA sequences. *Bioinformatics*, 19(Suppl 2):II206–II214, 2003.

51. R. Siddharthan, E.D. Siggia, and E. van Nimwegen. Phylogibbs: A Gibbs sampling motif finder that incorporates phylogeny. *PLoS Comput Biol*, 1(7):e67, 2005.

52. S. Sinha and M. Tompa. Ymf: A program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Res*, 31(13):3586–3588, 2003.

53. A.D. Smith, Z. Xuan, and M.Q. Zhang. Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9:128, 2008.

54. G.D. Stormo. DNA binding sites: Representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

55. G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. De Moor, P. Rouze, and Y. Moreau. A Gibbs sampling method to detect overrepresented motifs in the upstream regions of coexpressed genes. *J Comput Biol*, 9(2):447–464, 2002.

56. W. Thompson, E.C. Rouchka, and C.E. Lawrence. Gibbs recursive sampler: Finding transcription factor binding sites. *Nucleic Acids Res*, 31(13):3580–3585, 2003.

57. M. Tompa, N. Li, T.L. Bailey, G.M. Church, B. De Moor, E. Eskin, A.V. Favorov, M.C. Frith, Y. Fu, W.J. Kent, V.J. Makeev, A.A. Mironov, W.S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nat Biotechnol*, 23(1):137–144, 2005.

58. A. Valouev, D.S. Johnson, A. Sundquist, C. Medina, E. Anton, S. Batzoglou, R.M. Myers, and A. Sidow. Genome-wide analysis of transcription factor binding sites based on ChIP-seq data. *Nat Methods*, 5(9):829–834, 2008.

59. M.S. Waterman, R. Arratia, and D.J. Galas. Pattern recognition in several sequences: consensus and alignment. *Bull Math Biol*, 46(4):515–527, 1984.

60. H. Xu, C.L. Wei, F. Lin, and W.K. Sung. An HMM approach to genome-wide identification of differential histone modification sites from ChIP-seq data. *Bioinformatics*, 24(20):2344–2349, 2008.

61. Y. Zhang, T. Liu, C.A. Meyer, J. Eeckhoute, D.S. Johnson, B.E. Bernstein, C. Nussbaum, R.M. Myers, M. Brown, W. Li, and X.S. Liu. Model-based analysis of ChIP-seq (MACS). *Genome Biol*, 9(9):R137, 2008.

62. Z.D. Zhang, J. Rozowsky, M. Snyder, J. Chang, and M. Gerstein. Modeling ChIP sequencing in silico with applications. *PLoS Comput Biol*, 4(8):e1000158, 2008.

# 21

# APPROACHES AND METHODS FOR OPERON PREDICTION BASED ON MACHINE LEARNING TECHNIQUES

Yan Wang, You Zhou, Chunguang Zhou, Shuqin Wang, Wei Du,
Chen Zhang and Yanchun Liang

## 21.1 INTRODUCTION

The concept of operon appeared for the first time in the theory about a protein regulatory mechanism proposed by Jacob and Monod. An operon represents a basic transcriptional unit of genes in the complex biological processes of microbial genomes [1]. Therefore, operon prediction is one of the most fundamental and important research fields in microbial genomics [2].

Generally, an operon is a cluster of one or more tandem genes delimited by a promoter and a terminator, and its structure is shown in Figure 21.1. They usually have most of the same properties [3], which are very useful to identify an operon:

1. An operon consists of one or more genes on the same strand of a genomic sequence.
2. Intergenic distances within an operon are generally shorter than the distances of genes pairs without operons.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

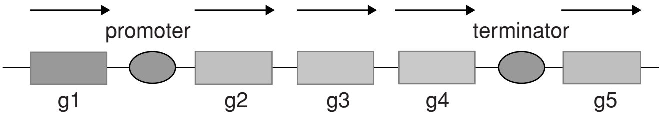


**Figure 21.1** The structure of an operon: g2, g3, and g4 compose an operon.

3. Generally, several genes of an operon have a common promoter and terminator, but the regions within an operon usually do not contain any promoter or terminator.
4. Genes in an operon usually have related functions, and most belong to the same functional category, such as a cluster of orthologus groups (COG) [4].
5. The genes in an operon as a functional unit tend to be found in more conserved gene pairs and more similar phylogenetic profiles.

In the past several years, numerous effective methods for operon prediction have been proposed. Through studying these methods, we find that:

1. Several operon databases currently are available on the Internet. These databases contain information with varying levels of reliability and different emphases, such as OperonDB [5], MicrobesOnline [6], ODB [7], RegulonDB [8], DBTBS [9], DOOR [10], and so on.
2. One can use one or more features of genome information to identify an operon. There are many different types of data to be used for operon prediction, including experimental data and computational data. The experimental data mainly includes gene expression data [11], metabolic pathways [12], and so on. The computational data includes genes data [13], genes functional categories [4], conserved gene pairs [14], phylogenetic profiles [15], and so on.
3. For the raw data, preprocessing often is necessary for data quality control and is important to eliminate noise that may be caused by instrument limitations and sampling errors.
4. There are many different kinds of methods for operon prediction, including the methods using training regulations and nontraining. The methods using training regulations include neural network [2], machine learning [16], and so on. In addition, comparative genomics [3] and genetic algorithm [17] are typical nontraining methods.

About the computational complexity, Zheng *et al.* [12] used theoretical evidence to show that the functional clusters extracting algorithm, which uses graph matching, costs exponential time. From the perspective of graph theory, the operon identification problem is a nondeterminister polinomial (NP) hard problem. In other words, there is no fast and rigorous algorithm for solving this problem.

## 21.2 DATASETS, FEATURES, AND PREPROCESSES FOR OPERON PREDICTION

### 21.2.1 Operon Datasets

It is of great significance for operon prediction to provide researchers detailed and effective information using query and statistical database-specific functions. Abundant operon databases are currently available on the Internet already. These databases contain information with varying levels of reliability and different emphases, such as OperonDB [18], MicrobesOnline [6], ODB [7], RegulonDB [8], DBTBS [9], DOOR [10], and so on. Here we will introduce six of the most popular operon databases related to different species as well as the results of different forecasting methods and manual experiments.

***21.2.1.1 OperonDB.*** OperonDB, first released in 2001, is a database containing the results of a computational algorithm for locating operon structures in microbial genomes. Initially, the method was used to analyze 34 bacterial and archaeal genomes and obtained more than 7600 pairs of genes that are highly likely to belong to the same operon. The sensitivity of this method is 30–50% for the Escherichia coli (*E. coli*) genome [16].

In 2008, OperonDB made great progress in both the size of the database and the operon finding algorithm [18]. It had grown from 34 genomes in its initial release to more than 500 genomes, and the method was redesigned to find operons with a higher accuracy. For each conserved gene pair, the probability that the genes belong to the same operon was estimated. The algorithm takes into account several alternative possibilities. One is that functionally unrelated genes may have the same order simply because they were adjacent in a common ancestor. Other possibilities are that genes may be adjacent in two genomes by chance alone or because of a horizontal transfer of the gene pair.

Until now, OperonDB is updated regularly and currently contains operon predictions for 843 bacterial and archaeal genomes, comprising the complete collection of finished prokaryotic genomes available from GenBank. This number will increase over time as more complete microbial genomes appear. All predictions can be downloaded in bulk, and the OperonDB software is available as free, open source software. It is available at http://operondb.cbcb.umd.edu.

***21.2.1.2 MicrobesOnline.*** The MicrobesOnline (http://www.microbesonline.org) is a resource for comparative and functional genomics that serves the scientific community for the analysis of microbial genes and genomes. The website was created by the Virtual Institute for Microbial Stress and Survival (VIMSS), a department of energy genomics; the Group Term Life insurance (GTL) project originally was developed to aid in the analysis of the Environmental Stress Pathway Project (ESPP) but has been open to the public since 2003 because of its broader utility to the scientific community.

Highlights of the MicrobesOnline website include operon and regulon predictions, a multispecies genome browser, a multispecies Gene Ontology (GO) browser, a comparative Kyoto Encyclopedia of Genes & Genomes (KEGG) metabolic pathway viewer, a bioinformatics workbench for indepth sequence analysis, and gene carts that allow users to save genes of interest for further study while they browse. In addition, it provides an interface for genome annotation, which is freely available to the scientific community [6].

***21.2.1.3 Operon Database (ODB).*** Operon Database (ODB) is created by Okuda S. of Kyoto University, Japan. ODB aims to collect the known operons in multiple species and to offer a system to predict operons by user definitions [7]. The current version of ODB contains about 2000 known operon information in more than 50 genomes and about 13,000 putative operons in more than 200 genomes. The database collects as many operons as possible from as many organisms as possible. This system integrates four types of associations: genome context, gene coexpression obtained from microarray data, functional links in biological pathways, and the conservation of gene order across the genomes. The integration of known literature-based information and genomic data are indicators of the genes that organize an operon in biology, and the combination of these indicators allows researchers to predict operons that are more reliable. As for the system to predict operons, ODB provides candidates of operons based on the constrict conditions of user's choice and provide its prediction accuracy. ODB is accessible at http://odb.kuicr.kyoto-u.ac.jp/.

In ODB, known operons are collected if a region transcribed as an operon is confirmed by experiments such as Northern hypridyzation or a set of genes are proved as an operon or a transcription unit in existing literature. Putative operons are defined by ortholog genes and the location on the genome. Operon prediction is provided by considering gene order on the genome of gene pairs obtained by combining "and" or "or" logic operations of the intergenic distance, intergenic step, pathway, coexpression, and ortholog parameters. The prediction accuracy is calculated from the operon pair, nonoperon pair, sensitivity, and specificity.

***21.2.1.4 RegulonDB.*** RegulonDB (http://regulondb.ccg.unam.mx/index.jsp) is a database that contains biological knowledge of the mechanisms that regulate the transcription initiation in *E. Coli* knowledge on the organization of the genes, and regulatory signals into operons in the chromosome [8]. The latest version of RegulonDB is 6.3.

RegulonDB provides the information of terminators, promoters, transcription factor (TF) binding sites, transcriptional factors-conformation, active and inactive transcription factor conformations, matrices_alignments, transcription units, operons, regulatory network interactions, Ribosome Binding Site (RBS), growth conditions, gene – product, small RNAs, and transcription start sites experimentally determined in the laboratory of Dr. Morett for users. There are also datasets through computational predictions, such as promoter predictions, operon predictions based on (intergenic) distances, TF binding site predictions, transcription factor predictions, riboswitch prediction, and attenuator prediction.

RegulonDB also provides tools for users such as genome zoom-tools, nebulon tool, RegulonDB overviews, regular sequence analysis (RSA) tools, GETools, RegulonDBTextpresso, promoter analysis tools, gene context tool, and genome reviews.

***21.2.1.5 DBTBS.*** DBTBS (http://dbtbs.hgc.jp) is a database of the upstream regulatory information of *Bacillus subtilis*. It recently was reorganized to show operons instead of individual genes as the building blocks of gene regulatory networks. DBTBS now contains 463 experimentally known operons as well as their terminator sequences if identifiable. In addition, 517 transcriptional terminators were identified computationally [9].

Compared with RegulonDB, which has been constructed as a powerful but slightly complicated relational database, DBTBS employs a simpler structure. It consists of two interrelated parts: one named “Regulated Operons,” which is a list of regulated genes. These genes are classified according to the functional category of their gene products by SubtiList [19]. The other is “Transcription Factors,” a compilation of experimentally characterized promoters in which the positions of known binding sites of transcription factors, including sigma factors, are given with links to PubMed reference information [20]. These positions are shown both in table format and in a gif image. In addition, there is a third main part named “Predictions.” This part contains four points; the first one is “predicted conserved hexameric motifs,” the second one is “predicted terminators,” the third one is “predicted operons,” and the last one is “predicted regulons.”

One notable feature of DBTBS is that overlapping binding sites can be perceived more easily (core consensus regions also are featured with color).

***21.2.1.6 Database of Prokaryotic Operons (DOOR).*** Database of prokaryotic operons (DOOR) is an operon database developed by the Computational Systems Biology Lab (CSBL) at the University of Georgia. It contains computationally predicted operons of all sequenced prokaryotic genomes. Currently, the DOOR database contains operons for 675 complete archeal and bacterial genomes and provides several search capabilities to facilitate easy access and utilization of the information stored in it [10].

The prediction algorithm was presented in 2007 by the same group [21], and it has been consistently the best at all aspects including sensitivity and specificity for both true positives and true negatives, and the overall accuracy reach is about 90% [22].

The database provides a search capability for users to find desired operons and associated information through multiple querying methods and to find operons that have similar composition and structure to a query operon. It also can predict cisregulatory motifs using motif-finding tools and includes operons for RNA genes. It is worth mentioning that the database provides a wiki page (OperonWiki) to facilitate interactions between users and the developer of the database.

### 21.2.2 Features

We can use one or more features of genome information to identify operons. Generally, using more types of features or properties may obtain a better effect if they are used properly. Therefore, the feature selection of the data is correlated significantly to the process outcome. Normally, there are many different types of data to be used for operon prediction, which mainly belong to the two categories of experimental data and computational data. The experimental data mainly include experimental evidence, such as gene microarray expression data, metabolic pathways, and so on. The computational data include genes data, intergenic spacing, genes functional categories and relations, sequence elements, conserved gene pairs, phylogenetic profiles, and so on. Operon prediction using features from experimental data can obtain high accuracies, but these methods generally are not applicable for newly sequenced genomes. The prediction using features from computational data cannot obtain an effect as good as experimental data, but they can be obtained without experiment so that these methods are easily applicable to newly sequenced genomes.

***21.2.2.1 Intergenic Spacing.*** The distance between open reading frames (ORFs) is a commonly used feature in the prediction of operons. Researches reported that if two adjacent genes on the same strand belong to the same operon, then their intergenic distance tends to be shorter. The intergenic distances between members of the same operon are relatively small as compared with those of genes that do not belong to the same operon. The property of intergenic distances is used frequently in operon predictions [23].

In Figure 21.2, we can see that the intergenic distances of most within operon (WO) pairs are between –10 bp and 20 bp, whereas the distances between transcription unit borders (TUB) pairs are more than 100 bp. Two adjacent genes, which belong to the same operons, were pairs of genes within operons; to the contrary, if they belong to different operons, then they were transcription unit borders.

#### *21.2.2.2 Conserved Gene Cluster*

*21.2.2.2.1 Conserved Gene Pairs.* The definition of conserved gene pairs is as follows: two adjacent genes A and B on the same DNA strand for which a homologous adjacent gene pair A′ and B′ can be found in another genome; like A is homologous to A′, B is homologous to B′. If the similarity between A and A′ and B and B′ are higher than the similarity between A and B, then the gene pair A and B is a conserved gene pair for the comparative genome.

In Figure 21.3, we find that in Figure 21.3a, the gene pair A and B is a conserved gene pair. In Figure 21.3b and Figure 21.3c, the gene pair of A and B is not a conserved gene pair. In Figure 21.3d, gene pair A and B is on the same DNA strand, but

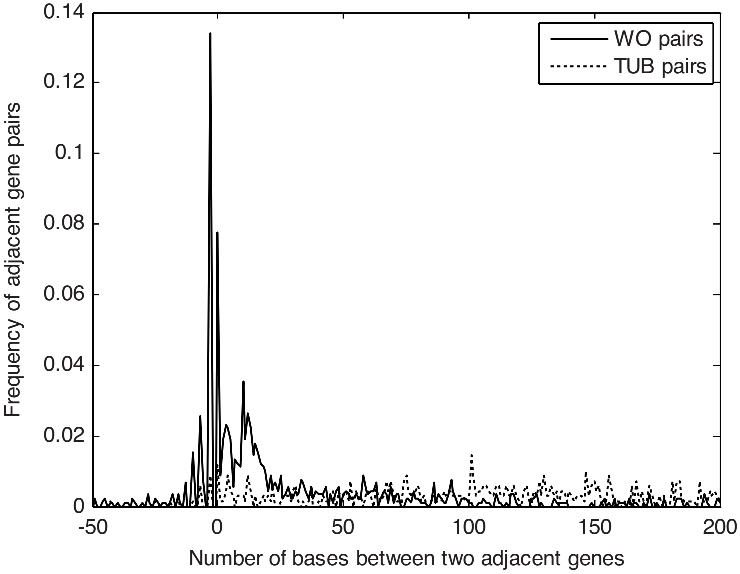


**Figure 21.2** Frequency distribution of different intergentic distances.

the similarity between A and B is higher than that between A and A′ or B and B′. Therefore, the gene pair of A and B is not a conserved gene pair. Operon prediction using conserved gene pairs first appeared in p [24].

*21.2.2.2.2 Phylogenetic Profile.* The phylogenetic profile of a protein is a binary string, each bit of which represents the presence or absence of the protein in the comparative genome. The phylogenetic profiles, which reflect similarity of homology, can provide certain information indicating gene functional categories and metabolism pathways. one or zero, respectively, represents that the protein appears or not in the comparative genome. The *n*th position of the binary string denotes whether the gene in the nth organism appears.

The key issue of using phylogenetic profiles is how to measure the corelation between two profiles. There are several profile distance measurements in some literatures.

**Hamming distance:** The Hamming distance was proposed when the concept of phylogenetic profile first was introduced [25]. It simply counts the number of different bits between two profile strings.

**Differential parsimony:** This measure, which was introduced in [26], calculates a differential parsimony in the historical evolution of two genes based on their phylogenetic profiles.

**Tree kernel:** With a powerful mathematical framework embedded, a tree kernel is proposed to analyze phylogenetic profiles [27].

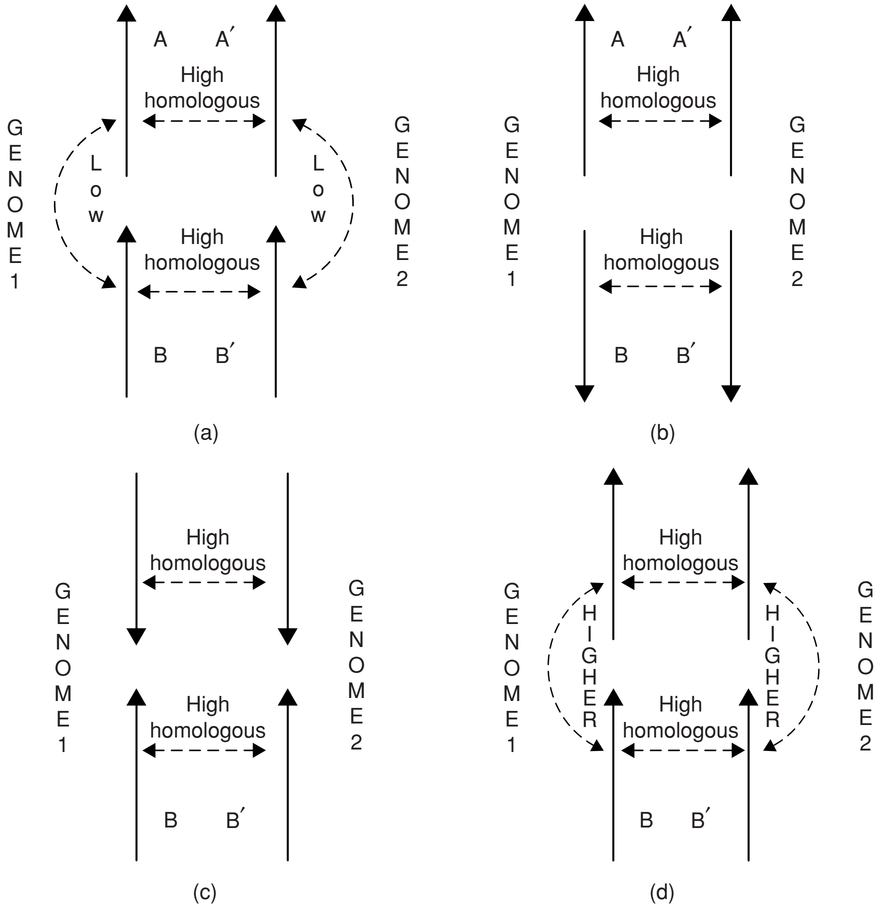


**Figure 21.3** Conserved gene pair and nonconserved gene pair.

### *21.2.2.3 Function Relation*

*21.2.2.3.1 COGs.* The COGs provide a framework for the analysis of evolutionary and functional relationships among homologous genes from multiple genomes. There are three major levels in COG function category [4]. The first level consists of four categories: (i) information storage and processing, (ii) cellular processes, (iii) metabolism, (iv) poorly characterized; the second level is a further refinement of the first level, which has more detailed categories; the third level is COG category numbers. Genes in an operon usually have related functions, and most of them belong to the same functional category, such as COG.

*21.2.2.3.2 GO.* GO provides three levels of biological functions: biological process, molecular function, and cellular component. It is known that genes in the same

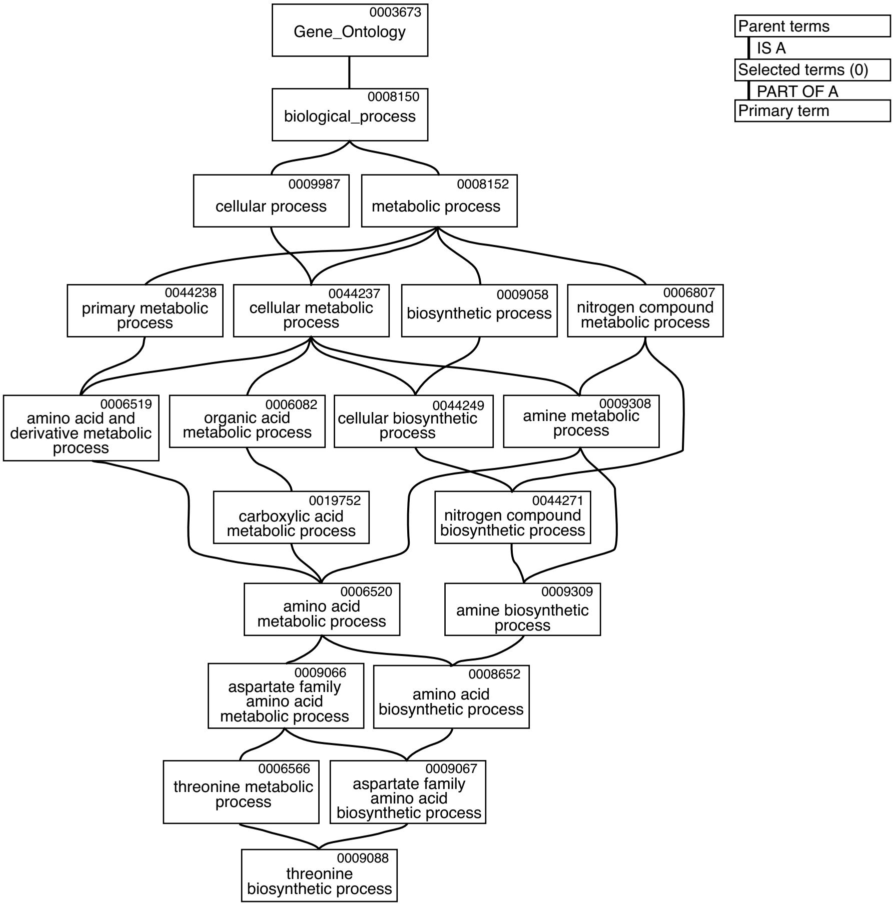
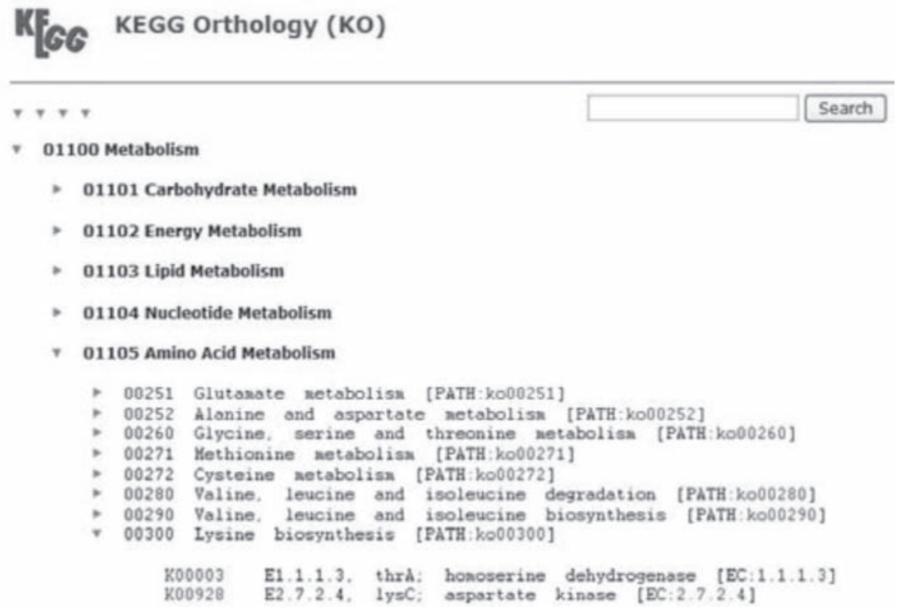

Parent terms
IS A
Selected terms (0)
PART OF A
Primary term
0003673 Gene_Ontology
0008150 biological_process
0009987 cellular process
0008152 metabolic process
0044238 primary metabolic process
0044237 cellular metabolic process
0009058 biosynthetic process
0006807 nitrogen compound metabolic process
0006519 amino acid and derivative metabolic process
0006082 organic acid metabolic process
0044249 cellular biosynthetic process
0009308 amine metabolic process
0019752 carboxylic acid metabolic process
0044271 nitrogen compound biosynthetic process
0006520 amino acid metabolic process
0009309 amine biosynthetic process
0009066 aspartate family amino acid metabolic process
0008652 amino acid biosynthetic process
0006566 threonine metabolic process
0009067 aspartate family amino acid biosynthetic process
0009088 threonine biosynthetic process

**Figure 21.4** A GO term (GO:0009088).

operon are involved in the same or similar biological processes; hence, GO information, in principle, should be helpful for operon prediction. We show a GO term in Figure 21.4 as follows.

*21.2.2.3.3 KEGG.* The KEGG pathways are organized into four general categories (level 1): Metabolism, Genetic Information Processing, Environmental Information Processing, and Cellular Processes. These categories are subdivided further into level 2 and level 3 pathways. The authors of KEGG have used KEGG Orthology (KO) terms to describe each pathway. Given a genome with annotated genes, one can assign KO terms to each gene and determine directly which pathway is involved. We show a KEGG term in Figure 21.5.



```
▾ 01100 Metabolism
    ▸ 01101 Carbohydrate Metabolism
    ▸ 01102 Energy Metabolism
    ▸ 01103 Lipid Metabolism
    ▸ 01104 Nucleotide Metabolism
    ▾ 01105 Amino Acid Metabolism
        ▸ 00251 Glutamate metabolism [PATH:ko00251]
        ▸ 00252 Alanine and aspartate metabolism [PATH:ko00252]
        ▸ 00260 Glycine, serine and threonine metabolism [PATH:ko00260]
        ▸ 00271 Methionine metabolism [PATH:ko00271]
        ▸ 00272 Cysteine metabolism [PATH:ko00272]
        ▸ 00280 Valine, leucine and isoleucine degradation [PATH:ko00280]
        ▸ 00290 Valine, leucine and isoleucine biosynthesis [PATH:ko00290]
        ▾ 00300 Lysine biosynthesis [PATH:ko00300]
              K00003   E1.1.1.3, thrA; homoserine dehydrogenase [EC:1.1.1.3]
              K00928   E2.7.2.4, lysC; aspartate kinase [EC:2.7.2.4]
```

**Figure 21.5** A KEGG term (K00003).

### *21.2.2.4 Sequence Elements*

*21.2.2.4.1 Short DNA Motifs.* To select the DNA motifs with the most discerning power (between operon pairs and boundary pairs), they have counted the number of occurrences for each DNA motif in the intergenic region of each gene pair. Dam *et al.* [21] found that some DNA motifs often seem to be associated with the interoperonic regions.

*21.2.2.4.2 Length Ratio Between a Pair of Genes.* The length ratio between a pair of genes is calculated as the natural log of the length ratio of an upstream gene and a downstream gene, or $L = \ln(l_i/l_j),\ \ j = i + 1$, where $l_i$ and $l_j$ are the length of the genes.

*21.2.2.4.3 Synonymous Codon Usage Biases (SCUB).* Within the standard genetic codes, all amino acids except Met and Trp are coded by more than one codon, which is called synonymous codons. DNA sequence data from diverse organisms show that synonymous codons for any amino acid are not used with equal frequency, and these biases are a consequence of natural selection during evolution. The synonymous codon usage biases (SCUB) also can be used to predict operons.

***21.2.2.5 Experimental Evidence.*** Genes in the same operon should have similar expression patterns, so several studies have used gene-expression data derived from microarray experiments to predict operons. With the development of experiment technology, gene expression microarray experiments become increasingly popular in eukaryote and prokaryote under different environmental conditions. This makes microarray experiment data available. Microarray data represent the expression intensities of genes. The genes in an operon are transcribed at the same level under many conditions, which means that the correlation coefficient of the expression value of genes in an operon should be equal to one in different microarray experiments. Hence, we can predict whether the genes are in an operon according to their correlation coefficient. Similar arguments have been verified by Sabatti *et al.* [28].

### 21.2.3 Preprocess Methods

As we discussed in previous sections, many raw data and features can be used for operon prediction. Therefore, preprocessing often is required for raw data cleaning and missing values imputation, and it is important to eliminate the noise that may be present in process measurements caused by instrument limitations and sampling artifacts. In the following sections, we will introduce three preprocess methods that are popular in this field.

***21.2.3.1 Log Likelihood.*** For a typical prediction problem with many different types of data, the relations between them must be identified. The Log-likelihood formula is then defined as follows:

$$\mathrm{LL}(\mathrm{WO}|d(g_a, g_b)) = \log \frac{P(d(g_a, g_b)|\mathrm{WO})}{P(d(g_a, g_b)|\mathrm{TUB})} \tag{21.1}$$

where $P(d(g_a, g_b)|\mathrm{WO})$ and $P(d(g_a, g_b)|\mathrm{TUB})$ are the anterior probability that a specific relation $d(g_a, g_b)$ could be observed in WO genes pairs or TUB genes pairs, respectively, and $g_a$ and $g_b$ are the property values observed for two adjacent genes. $\mathrm{LL}(\mathrm{WO}|d(g_a, g_b))$ is the logarithm likelihood score, which expresses the probability of an adjacent gene pair belonging to the same operons. Adjacent gene pairs with higher log-likelihood scores are more likely to be WO pairs [2].

***21.2.3.2 Entropy.*** Wang *et al.* [29] use the log-energy entropy to preprocess the intergenic distances of WOs and TUBs by using Equation (21.2).

$$d_{\text{entropy}} = \log(d^2_{\text{distance}}) \tag{21.2}$$

Wang *et al.* [30] propose a local-entropy-minimization method for preprocessing intergenic distance. Intergenic distance frequencies of known WO pairs and of known TUB pairs are examined in the three aforementioned genomes. To obtain the pair-scores according to the intergenic distances, all intergenic distance entropies are calculated by using Equation (21.3):

$$E(d) = -p(d)\log[p(d)] - [1 - p(d)]\log[1 - p(d)] \tag{21.3}$$

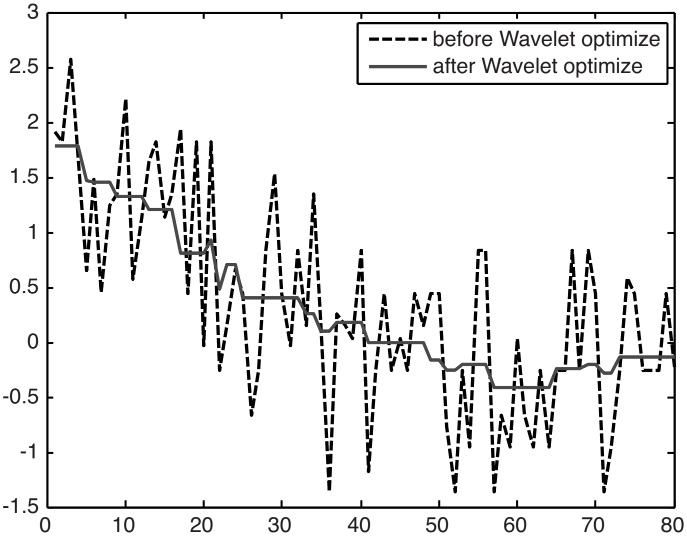


**Figure 21.6** The effect of the wavelet transform optimization.

***21.2.3.3 Wavelet Transform.*** The wavelet transform first was introduced into bioinformatics research in 1996 and attracted abroad attention immediately. The wavelet transform differs from the Fourier transform in that it allows the localization of a signal in both time and frequency.

Du *et al.* [31] use the wavelet transform to optimize the log-likelihood scores of intergenic distances, conserved gene pairs, and phylogenetic profiles. Because log-likelihood scores are calculated based on probability measurements, they are likely to contain much noise information. So the authors deal with these log-likelihood scores using wavelet denoising and wavelet compression. The effect of the wavelet optimization is shown in Figure 21.6.

## 21.3 MACHINE LEARNING PREDICTION METHODS FOR OPERON PREDICTION

In this section, we will discuss the machine learning techniques involved in operon prediction. There are many different kinds of methods for operon prediction, including the methods using training regulations and nontraining. The methods using training regulations include the hidden Markov model (HMM), linkage-clustering, Bayesian classifier, Bayesian network, support vector machine (SVM), neural network (NN) and so on. Comparative genomics and genetic algorithm are typical nontraining methods. The methods using training regulations can obtain high sensitivities and accuracies but with less adaptability. On the other hand, the adaptability of the nontraining methods is good but cannot estimate the important degree of data in prediction. In the following, each method is described, and the recent approaches are highlighted.

### 21.3.1 Hidden Markov Model

The hidden Markov model (HMM) is a statistical model that assumes that the modeled system is a Markov process with an unobserved state. An HMM can be considered the simplest dynamic Bayesian network. Hidden Markov models especially are known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges, and bioinformatics.

Here is an example using *E. coli* oligonucleotide microarrays to assay transcript expression of both ORFs and intergenic regions. Then use hidden Markov models to analyze this expression data and estimate transcription boundaries of genes [32].

The transcript expression data is obtained using Affymetrix (Santa Clara, CA) high-density oligonucleotide arrays. Each array contains 295,936 spots or probes. Every *E. coli* ORF is assayed by a set of perfect match (PM) and mis-match (MM) probe pairs, and each intergenic region, at least 40 base pairs in length, is assayed in both orientations by a set of probe pairs.

Considering a set of $T = 15$ probes assaying an intergenic region upstream of an ORF. Let $\mathrm{O} = \mathrm{O}_1, \mathrm{O}_2, \ldots, \mathrm{O}_\mathrm{T}$, where $\mathrm{O}_\mathrm{j}$ is the correlation coefficient of the expression vector of probe j with the expression vector $\theta$ of the ORF, and let $\mathrm{I} = \mathrm{I}_1, \mathrm{I}_2, \ldots, \mathrm{I}_\mathrm{T}$, where $\mathrm{I}_\mathrm{j}$ equals 1 if probe j assays the 5' untranslated region (UTR) of the gene and $\mathrm{I}_\mathrm{j}$ equals 0 otherwise. If we knew the exact 5'UTR for the gene, then we would know which of the T probes assayed the gene transcript and which did not. However, we do not know which probes assay UTR regions for the gene; rather we view this information as "hidden," and the goal is to determine the sequence I that maximizes the joint probability P(O, I). Furthermore, to expect $\mathrm{P}(\mathrm{I}_\mathrm{j} = 1|\mathrm{I}_{\mathrm{j}-1} = 0)$ to be near zero given that probe j-1 upstream of an ORF does not assay part of the gene's 5'UTR, we do not expect probe j farther upstream to assay part of the gene's 5'UTR. This dependency on previous "states" motivates our use of HMMs.

Figure 21.7 shows a simple two-state HMM that can be characterized by three sets of parameters, $\lambda = (A, B, \pi)$. Initial probabilities are represented by $\pi = \{\pi x\}$, where $\pi x$ is the probability of starting in state x. Transition probabilities are represented by $A = \{axy\}$, where $axy$ is the probability of being in state y given that the prior state was x. Emission probabilities are represented by $B = \{bx(r)\}$, where $bx(r)$i s the probability of generating or emitting the correlation coefficient $-1 \leq r \leq 1$ in state x. The goal is to determine the most likely transcript boundary of a gene given the correlation data upstream or downstream of the ORF.

Using a dynamic programming approach, the VITERBI Algorithm can determine this optimal state sequence. This approach, however, relies on having a model for the HMM with appropriate parameters for $A$, $B$, and $\pi$. The emission probabilities $B$, may use the two smoothed distributions. The initial probabilities $\pi$and transition probabilities $A$ train the HMM from a set of observation sequences using the

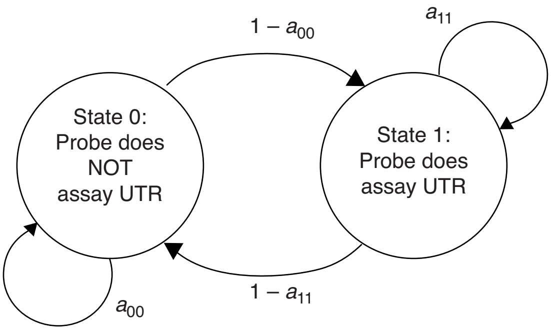


**Figure 21.7** A two-state HMM. State 0 corresponds to a probe that assays a region, which is not part of the gene's UTR. State 1 corresponds to a probe that assays a region, which is part of the gene's UTR. The arrows represent the transitions between states.

segmental K-MEANS algorithm so that $A$ and $\pi$ are chosen to maximize $P(O, I\,|\lambda)$. Then implement two sets of HMMs: one for operon identification and the other for 5'UTR identification.

When extending the analysis to incorporate the expression of intergenic regions using the HMM approach, the results correctly identified operon elements with 99% specificity and 63% sensitivity. These results are based on 115 positive examples of documented operon elements with intergenic regions greater than 40 base pairs and where we can observe at least minimal transcript expression of the ORFs and from 115 randomly chosen adjacent misoriented gene pairs as negative examples. Furthermore, an additional 227 new operon elements were identified from the HMM analysis.

The complexity of the hidden Markov model is related to the order of the Markov models, for example, the order 0 Markov models, order 1 Markov models, and so on until order m Markov models. An order 0 Markov model has no "memory," and it is equivalent to a multinomial probability distribution. An order 1 (first-order) Markov model has a memory of size 1.

### 21.3.2 Linkage Clustering

Linkage clustering is a hierarchical clustering method using linkage criteria as the measure of dissimilarity.

1. **Single linkage clustering**
   Single linkage clustering specifies the distance between groups as the distance between the closest pair of objects in different groups. See Figure 21.8 as an example.
2. **Complete linkage clustering**
   Complete linkage clustering specifies the distance between groups as the distance between the farthest pair of objects in different groups. See Figure 21.9 as an example.

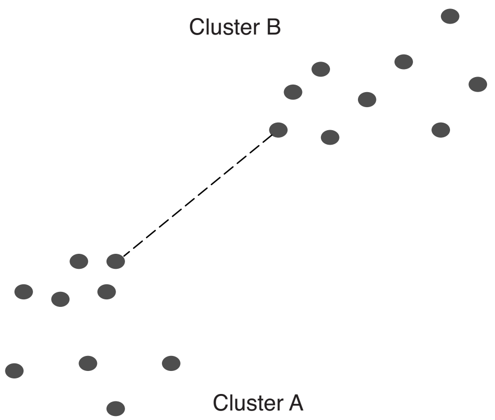


**Figure 21.8** Single linkage clustering.

3. **Average linkage clustering**
   Average linkage clustering specifies the distance between groups as the average distance between the pair of objects in different groups. See Figure 21.10 as an example.

The average linkage hierarchical clustering algorithm has been employed to cluster genes based on the similarity of their functional linkage profiles that use the comparison metric depending on a centered correlation coefficient. The relationships among gene expression patterns are represented by a tree, and the branch lengths of the tree represent the degree of similarity between the gene profiles. The *trp* operon were contained mostly in one branch of the gene tree. The clustering reflects both bacterial operon organization and close chromosomal proximity of genes of related function [33].

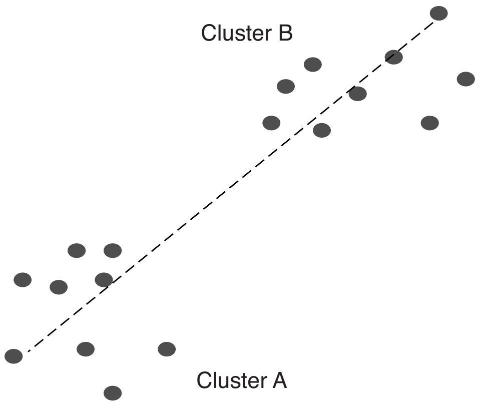


**Figure 21.9** Complete linkage clustering.

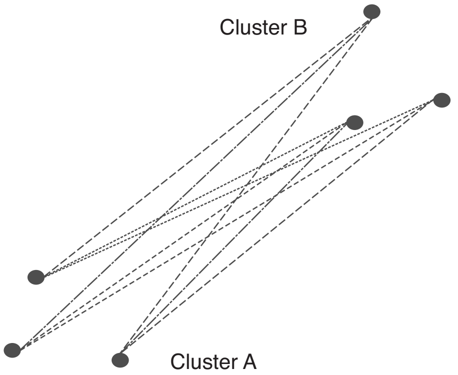


**Figure 21.10** Average linkage clustering.

The sequential implementations of hierarchical clustering have a computational complexity between O($N^2$) and O($N^3$) [33].

### 21.3.3 Bayesian Classifier

The Bayesian classifier is based on Bayes' theorem. Its training process is a statistical learning process, resulting in a corresponding classifier. The basic model is the naïve Bayesian classifier, which is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. The formula of naïve Bayes is presented as follows:

$$\Pr(O|D) = \frac{\Pr(D|O)\Pr(O)}{\Pr(D)} \tag{21.4}$$

Where O is a random variable indicating whether a candidate actually is an operon, and D represents the data available to make its determination.

The core of the task is to estimate the likelihood of the data of interest given the two possible outcomes for O. Using the naïve Bayes method, it makes the assumption that features are independent of one another given the class and therefore make the following approximation:

$$\Pr(D|O) \approx \prod_i \Pr(D_i|O) \tag{21.5}$$

where $D_i$ is the $i_{\mathrm{th}}$ feature.

We will use a specific example to interpret how to predict operons based on Bayesian classification. Before representing the details, let us introduce the background of the problem first. The operon status of selected genes was derived from RegulonDB with the following revisions: (i) leader peptides were removed from

**Table 21.1 Comparison of sensitivity and specificity of the different operon prediction methods**

| Method | Sensitivity | Specificity |
|---|---|---|
| Correlation | 0.82 | 0.70 |
| Distance | 0.84 | 0.82 |
| Distance and correlation | 0.88 | 0.88 |

the operon set because of transcriptional attenuation within the operon, which may disrupt the coexpression pattern within the operon; (ii) operons with significant secondary promoters were deleted; (iii) new operons were added based on literature data. This is the operon training set.

Then examine what extent the microarray data set contributes to the prediction of operons beyond the prediction based on gene distance. To achieve this goal, they used the Bayesian framework, which makes it particularly easy to update the current knowledge on a pair of genes based on novel information.

Defining an overall noise considering all DOPkj across all experiments: Noise $SD = \text{mad}_{kj}(|\text{DOP}_{kj}|)/\sqrt{2}$. The ingredients of the Bayesian classification procedure are the prior distributions and the likelihood $f(r|\text{OP})$ and $f(r|\text{NOP})$ (distribution of correlation given the operon status). Estimating the functions $f(r|\text{OP})$ and $f(r|\text{NOP})$ with the smooth densities is represented based on the previously described collection of 604 operon pairs (OPs) and 151 nonoperon pairs (NOPs). As two different specifications of the prior distribution were considered, there are two sets of posterior probabilities; one is based only on the expression correlation, Post(OP|$r$); the other is based on both the distance and the expression correlation.

Then evaluate the correct classification rates for the 604 known OPs and 151 NOPs obtained when classifying as OPs all Potential Operon Pairs (POPs) for which (i) Post(OP|$r$) $>$ 0.5, (ii) Post(OP|$d$) $>$ 0.5, or (iii) Post(OP|$r$,$d$) $>$ 0.5. To avoid underestimating the error rate, a leave-one-out cross-validation procedure was used so that each POP, in turn, is excluded from the training set; the likelihoods are reestimated, and the status of the POP is predicted based on the newly evaluated decision boundary. The calculated results of the percentage of correctly classified operons (sensitivity of the operon prediction rule) and correctly classified nonoperons (specificity of the operon prediction rule) are shown in Table 21.1.

As a benchmark, the sensitivity and specificity of a uniform random classification of an operon and a nonoperon are equal to 0.5. Hence, using the correlation of expression values across microarray experiments produces a 64% increase in sensitivity and a 40% increase in specificity. This classifier represents one of the current standards for operon prediction; the fact that a comparable performance could be obtained with array data is an indication that correlation between expression levels indeed contains a considerable amount of information.

Then apply this prediction based on both microarray data and distance to the completely POP set. The POPs in the *E.coli* genome (total 3024) comprise the collection of all pairs of adjacent genes that are transcribed in the same direction (including OPs, NOPs, and POPs of unknown status). The results of these predictions are presented graphically in an expression correlation map of the entire *E. coli* genome. The structure of the map is illustrated in Figure 21.11.

The purpose of a Bayesian model selection approach based on the model likelihood is to tell us whether the increased complexity of a model with more parameters is justified by the data. However, the number of free parameters in a model is only the simplest possible notion of complexity, which is called input complexity and is denoted by C0. A much more powerful definition of model complexity was given by Spiegelhalter *et al.* [34], who introduced the Bayesian complexity, which measures the number of model parameters that the data can constrain:$C_b = -2(D_{\text{KL}}(p, \pi) - \hat{D}_{\text{KL}})$, where $D_{\text{KL}}(p, \pi)$ is the Kullback–Leibler (KL) divergence between the posterior and the prior; The KL divergence measures the relative entropy between the two distributions; $\hat{D}_{\text{KL}}$ is a point estimate for the KL divergence.

This operon prediction approach is from one paper written by Sabatti *et al.* [28], which explains it in more detail.

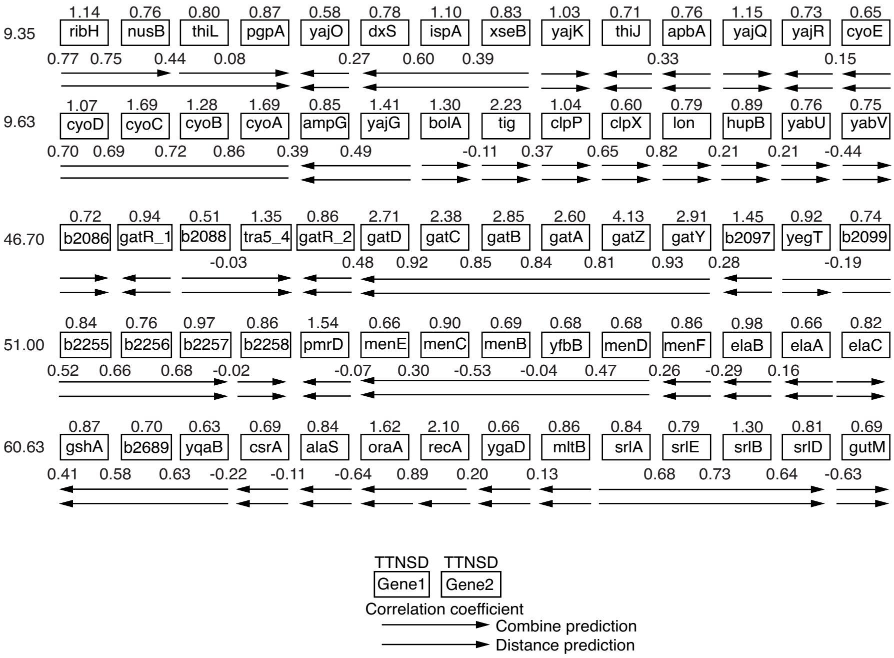


**Figure 21.11** An example of a gene map. Each box represents a gene, the number in which is the Total-To-Noise-Standard-Deviation (TTNSD) value for the corresponding gene. Between the genes transcribed in the same direction is the correlation coefficient given. The first arrow represents the predicted operon structure when both distance and microarray data are taken into account. The second arrow represents the operon structure as it is predicted by distance alone. On the left, an approximate minute count is given.

### 21.3.4 Bayesian Network

A Bayesian network is used to represent the joint probability distribution of a set of random variables that exploits the conditional independence relationships among the variables, often greatly reducing the number of parameters needed to represent the full joint probability distribution. Also, the Bayesian network provids a powerful and natural way to represent the dependencies that do exist.

Bockhorst *et al.* proposed a method with a Bayesian network to link many kinds of observations such as spacer sizes, expression profiles, and codon usage [35]. Although some methods like Bockhorst's use correlations between only adjacent genes on a single DNA strand, a pair of genes that are not immediately next to each other can be an operon pair when the members of the operon are more than two. Therefore, we mainly introduce a method proposed by Shimizu *et al.* for the operon prediction, which uses correlations between not only adjacent but also distant genes on a DNA strand [36].

As we know, any pair of genes that are adjacent to a single DNA strand can be divided into two groups, OP or NOP. The prediction is to divide all pairs of adjacent genes into predicted OPs and predicted NOPs. It is evaluated whether it is consistent with the operon structures determined by biological experiments. Sensitivity (correctly predicted OP/known OP) and specificity (correctly predicted NOP/known NOP) are used as evaluation criteria. Then a Bayesian network model can be formulated like in Figure 21.12 that is assumed to generate correlation of every gene pair. $z_{i,j}$ is a hidden variable that takes 0 or 1, corresponding when the pair of gene$_i$ and gene$_j$ is NOP or OP, respectively. $r_{i,j}$ is the correlation coefficient between gene$_i$ and gene$_j$, which is assumed to be generated randomly depending on $z_{i,j}$. This generation process, $p(r_{i,j} \,|z_{i,j})$, is determined beforehand using biologically known operon

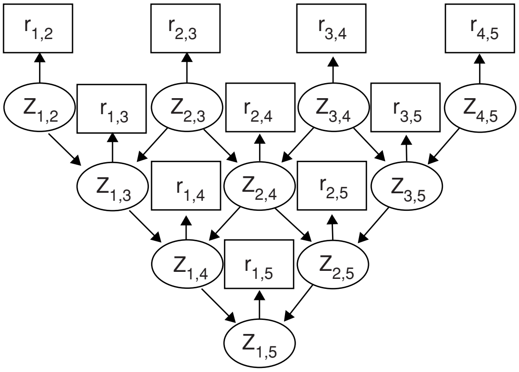


**Figure 21.12** A Bayesian network model (left). Squares represent observable variables ($r_{i,j}$), and circles represent hidden variables ($z_{i,j}$). An arrow stands for dependence. Hidden variables have a hierarchical dependence structure. For example, if a pair of distant genes is OP, then any pair of genes between the two genes is always OP. $r_{i,j}$ depends on the corresponding hidden variable, $z_{i,j}$, which means that the distribution of $r$ is dependent on whether the pair is OP or NOP.

structures. The problem is to estimate the posterior distribution of $z_{i,j}$ for a given set of $r_{i,j}$ calculated from microarray data. They used variational Bayes method to approximate the posterior probability, and the the new method is more effective than the naive one according to some experiments.

Because the Bayesian network model has different structures, there is a study to show that subsampling with network induction (SSML) cannot guarantee monotonically faster inference with increasing network approximations; this results because the network structure induced from approximate data (sampled from the original network) has a high variance. In contrast, with edge deletion (ED), the trade-offs of accuracy for faster inference are guaranteed to be monotonic. So, for several large Bayesian Networks (BNs), ED can create approximate networks with order-of-magnitude inference speed-ups with relatively little loss of accuracy [37].

### 21.3.5 Support Vector Machine

SVM, based on the statistical learning theory, can be used to solve a target classification problem. It provides a better way to deal with several biological problems, such as protein interactions and the accurate identification of alternatively spliced exons [38].

With respect to the task of operon prediction, gene pairs fall into two classes: WO pairs that are treated as positive data and TUB pairs that are treated as negative data. The SVM should define a hyper plane to classify them $yi(wTx_i + b) \geq 1$. The margin, denoting the distance of the hyper plane to the nearest of the WO and TUB pairs, is $2/||\omega||$ and evaluates the classification ability of SVM [39]. For optimal resolution of the data, the SVM analysis should arrive at an optimized hyper plane with a maximal margin. It can be expressed as the following optimization problem:

$$\begin{cases} \min \frac{1}{2} w^T w \\ y_i(w^T w + b) \geq 1, \ \forall i \end{cases} \tag{21.5}$$

By introducing Lagrange multipliers $\alpha$i, the optimization problem converts into a dual form that is a quadratic programming (QP) problem. Once a SVM is trained, the decision function can be written as follows:

$$f(x) = \operatorname{sgn}\left( \sum_{i=1}^{N} y_i \alpha_i x x_i + b \right) \tag{21.6}$$

In nonlinearly separable cases, the SVM technology introduces the slack variable $\{\xi_i\}_{i=1}^{N}$ to the definition of hyperplane.

SVM maps the input variable into a high-dimensional feature space with a kernel function $K(x_i, x_j)$. The decision function can be defined as

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{N} y_i \alpha_i K(xx_i) + b\right) \tag{21.7}$$

Wang *et al.* [29] apply the least-square support vector machine (LS-SVM) to operon prediction of *E. coli* with different combinations of intergenic distance, gene expression data, and phylogenetic profile.

Introduced to the problem of operon prediction, gene pairs can be classified into two classes: WO pairs that take the role of positive data (class label +1) and TUB pairs that take the role of negative data (class label –1). The vectors are the features of the preprocessed data, which is Log-energy entropy intergenic distance, Hamming distance of phylogenetic profiles, and denoised Pearson correlation coefficient of the gene-expression data.

Operons are predicted in the *E. coli* K12 genome by LS-SVM, with 300 WO pairs and 200 TUB pairs as the train sets and the other as the test sets. The best results of the prediction were yielded from the combination of the entire feature as vectors in the kernel of linear type, of which sensitivity, specificity, and accuracy were 90.73%, 93.54%, 92.34%, respectively. By comparing the results with library for support vector machines (Lib-SVM), joint predection of operons (JPOP), and Operon Finding Software (OFS), it showed the LS-SVM method got the better results.

Zhang *et al.* [39] used the SVM approach to predict operons at the genomic level. Four features were chosen as SVM input vectors: the intergenic distances, the number of common pathways, the number of conserved gene pairs, and the mutual information of phylogenetic profiles. The analysis reveals that these common properties are indeed characteristic of the genes within operons and are different from that of nonoperonic genes. Jackknife testing indicates that these input feature vectors, employed with radial basis function (RBF) kernel SVM, achieve high accuracy.

The operons in *E. coli* K12 and *B. subtilis* are used commonly as benchmarks in operon prediction. The SVM system used in this work is Lib-SVM. In this program, gene features can be set as the input vectors for SVM.

All genes in a target genome can be grouped into clusters based on their orientation relative to flanking genes, Any two adjacent genes can be designated as a gene pair, regardless of their orientation.

The process of operon prediction includes the following steps:

1. Cluster the genes in a genome
2. Eliminate genes that are located in single-gene clusters
3. Break down multigene clusters into potentialWO pairs
4. Extract features of candidate WO pairs
5. Identify true WO pairs and eliminate TUB pairs from candidate WO pairs

The SVM system accuracy was evaluated by two sets of benchmark operons: *E. coli* and *B. subtilis*. Both sensitivity and specificity were higher than 80%, which means the SVM achieved a good balance between the two measurements.

The complexity of the support vector machine is related to the type of kernel function, for example, polynomial kernel, linear kernel, RBF kernel, Gaussian radial basis function, hyperbolic tangent function, and so on. The complexity of SVM with nonlinear function is higher than that with linear function.

### 21.3.6 Artificial Neural Network

Artificial neural network (ANN) is a nonlinear dynamic computational model that tries to simulate the structure and/or functional aspects of biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases, an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. It has many advantages such as self-learning, self-adapting, and self-organizing capabilities. The neural cells in neural work are an adaptive process unit, and it stores knowledge in the linkage of cells. The weight in the linkage is modified by some kinds of rules such as the Hebb rule, $\delta$ learning rule, and Widrow–Hoff learning rule.

Today, there are many kinds of neural network models such as the perceptron model, back-propagation (BP) network model, self-organizing map (SOM) model, recurrent network model, and hybrid network model.

The most commonly used model is BP network. It is a multilayer network, which consists of input layer, output layer, and several hidden layers. BP network is a forward network, and the training algorithm is a supervised back–forward algorithm. It uses the gradient descent method to modify the weight of network and to minimize the mean squared error (MSE) of network outputs and actual datasets. The principle of the gradient descent method is to initialize a group of weights first and then calculate the gradient of mean square error to weight and modify the weight; the algorithm terminates when MSE minimizes to a certain limit. In detail, the algorithm can be defined as follows.

Every node executes a nonlinear proceeding via weight input and gets its dynamic outputs. The most commonly use sigmoid function is expressed as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{21.8}$$

For one process unit, if there are $K$ training samples $\{x^k\}$, the corresponding output is $\{T^k\}$, the weight of network is $W$, the MSE of this process unit is$\varepsilon$, then

$$\varepsilon = \frac{1}{K} \sum_{K=1}^{K} (W \times X^k - T^k)^2 \tag{21.9}$$

So $\varepsilon$ is a function of weight $W$. Then it can modify the weight by the gradient descent method:

$$W^* = W - \eta \Delta \varepsilon \tag{21.10}$$

where $\eta$ is the step to control the pace of modifying weights, and $\Delta\varepsilon$ is the gradient of $W$.

A neural network is a uniform machine learning method, and it is successful when implemented into operon prediction. There are many methods to predict operons based on the neural network. The most famous program is JPOP [2], which classifies each pair of consecutive genes as an "operonic" or an "nonoperonic" boundary based on their intergenic distance, similarity between their phylogenetic profiles, and relatedness of their annotated functions from COGs [4]. Each of these sets of supporting data is integrated using a NN to generate operon predictions.

### 21.3.7 Genetic Algorithms

A genetic algorithm (GA) is a search technique that has been used frequently in calculations to find approximate solutions to optimization and search problems. The primary steps are as follows:

1. **Initialization**
   The encoding of individuals and the population size in an initial population are considered depending on the actual problems. The common solution is represented in binary as strings of 0s and 1s.
2. **Selection**
   During each successive generation, a proportion of the existing population is selected to breed a new generation using the fitness function, which depends on the specific problem.
3. **Reproduction**
   Given a pair of "parent" solutions, the "child" solutions are generated with crossover and mutation.
   1) **Crossover**:
      The value of probability for crossover is important, and one choise is around 0.7. Crossover is performed by selecting a random gene along the chromosomes and swapping all genes after that point. See Figure 21.13 as an example.

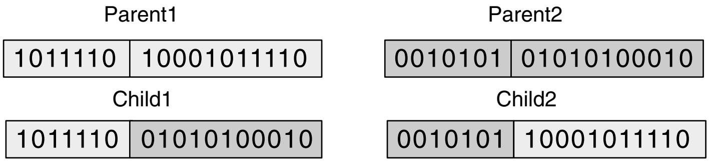


**Figure 21.13** Crossover.

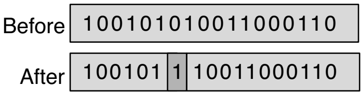


**Figure 21.14** Mutation.

2) **Mutation:**
   Mutation is performed with a very low probability because the mutation rarely appears in nature, and it is commonly 0.001. See Figure 21.14 as an example.
3) **Termination:**
   The generational process is repeated until a termination condition is satisfied. There are some common terminating conditions such that a solution is found that satisfies minimum criteria, the number of generations exceed a fix value, and so on.

A fuzzy guided genetic algorithm has been employed to predict operon. This method used a genetic algorithm to evolve an initial population, which presents a putative operon map of a genome. Each putative operon is scored using intuitive fuzzy rules, and a high accuracy of prediction can be obtained [17]. A modified genetic algorithm using the fitness function based on four kinds of genome information has been employed to predict an operon. The genome information includes intergenic distance, COG gene functions, metabolic pathway, and microarray expression data [30].

The computational complexity of the genetic algorithm is O $(t \times m \times n)$, where $t$ is the generation, $m$ is the population size, and $n$ is the objective size [40].

### 21.3.8 Several Combinations

All methods we mentioned use only one algorithm through machine learning to predict operons. Operons also can be predicted through combining several algorithms. This method, which is introduced as an example of algorithms combination, predicted operons in *Pyrococcus furiosus* by combining the results from three existing algorithms using a NN. These algorithms use intergenic distances, phylogenetic profiles, functional categories, and gene-order conservation in their operon prediction.

Because no genome-wide operons in *P. furiosus* have been determined experimentally, the program was benchmarked using operons from *E. coli* and *B. subtilis*, which have been validated experimentally. The true positive (TP) set in *E. coli* are the transcriptional unit gene pairs extracted from the RegulonDB database. A negative dataset is generated as follows: two adjacent genes in the same direction are considered as a true negative (TN) gene pair if they are not transcriptionally coexpressed (*i.e.*, not in the same transcriptional unit). In addition, only gene pairs with confidence measures from all three prediction programs can be considered when generating the TP and TN sets.

The details of the algorithm beginning with pathway assignment is described as follows.

Genes within the same operon generally encode proteins that function in the same metabolic pathway or biological process. As such, scores have been generated from the pathway information collected from the KEGG database [41]. The KEGG (prokaryotic) pathways are organized into four general categories (level 1): Metabolism, Genetic Information Processing, Environmental Information Processing, and Cellular Processes. These categories are subdivided further into level 2 and level 3 pathways. The authors of KEGG have used KO terms to describe each pathway.

Once the ORFs are assigned KO annotation, the KEGG pathways can be inferred directly. A KEGG pathway score of 1, 2, or 3 was assigned to a gene pair if they share the same level 1, level 2, or level 3 pathway, respectively. The higher the score, the higher the chance the two gene products are in the same pathway, and hence, it is more likely that the two genes belong to the same operon.

Another input to the NN-based predictor is an intergenic distance-based log-likelihood score defined by Equation 21.11. This score for a gene pair is computed as the log ratio between the probability that their distance belongs to the distance distribution of the TP set and the probability that distance belongs to the distance distribution of the TN set.

$$\mathrm{LL}(d(g_a, g_b)) = \ln \frac{P(d(g_a, g_b)|\mathrm{TP}_{\mathrm{genepair}})}{P(d(g_a, g_b)|\mathrm{TN}_{\mathrm{genepair}})} \tag{21.11}$$

**Score Normalization**

Normalization of the confidence scores of the three prediction programs is necessary to ensure that the dynamic range of the individual programs does not influence the performance of the ANN. For each program, the prediction confidence measure for each gene pair was extracted and normalized to between 0 and 1, where a value >0.5 indicates that the corresponding gene pair belongs to the same operon. The GO similarity score, KEGG pathway scores and log-likelihood score, of intergenic distance also were normalized linearly into the range [0, 1].

**Neural Network Training**

The idea of ANN is to train a set of parameters to give a desired output target ($t$), for a given input data ($x$). In this approach, the authors present the confidence measures from each of the three prediction programs, $x = [xi]$ for $i = 1, 2, 3$ to a feedforward network architecture (Figure 21.15). Various combinations of GO similarity, KEGG pathway, and intergenic distance scores also were tested as additional inputs into the NN-based predictor. The desired output target is 0/1 {1 = "gene pair in an opero", 0 = "gene pair not in an operon"}. The training algorithm optimizes the weights $W = [wi]\mathrm{T}$ and a bias, $b$, of the network during the training (supervised learning) phase to minimize the error between the network output, $a$, and the desired output, $t$, on the

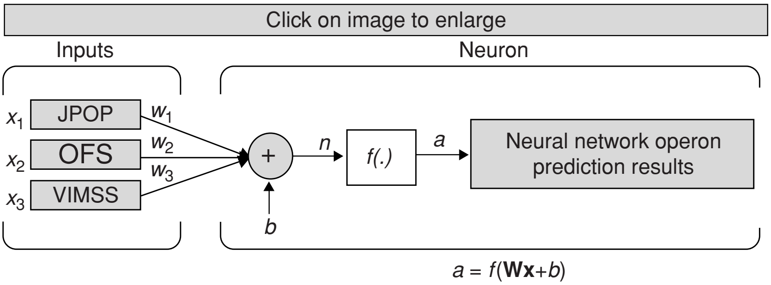


**Figure 21.15** Schematic illustration of a one-layer NN architecture with three inputs from existing programs. The confidence values $x_i$ of each operon prediction program are inputs into a neuron consisting of a summation unit and a transfer function, $f$, to produce an output $a$.

training data. The network architecture parameters are (i) the transfer function ($f$), (ii) the number of neurons, and (iii) the number of layers.

This approach for operon prediction is proposed by Tran *et al.* [42], which is described herein as integrating the strengths of existing prediction algorithms that use various sequence features such as codon usage and intergenic distance, conserved gene-order, and phylogenetic profiles of genes.

## 21.4 CONCLUSIONS

Operons are critical to the reconstruction of regulatory networks at the whole genome level. They can provide highly useful information for characterizing or constructing the regulatory network in a microbial genome. A better operon prediction result is more powerful for predicting regulon, pathway, and regulatory network.

In this chapter, we introduced six operon databases with comprehensive operon information, including dataset size, data source, prediction method if necessary, and so on. We also generated the features that are most popular in operon prediction methods today and the general methods to prepare the data. Then eight machine-learning algorithms for operon prediction are described for the readers who are interested in computational algorithms solving the problems of operon identification.

We hope this chapter is helpful for readers to learn about operon prediction quickly and is convenient to use it as a handbook. Furthermore, there are still lots of work that needs to be done in the operon prediction research field in the future, such as how to improve the prediction accuracy and integrate operon prediction software into some platforms for comparative analyses of prokaryotic genomes, and so on.

Because of the author, there might be some errors in this chapter. Please contact us and we will correct it in the future. For more detail about topics covered in this chapter, please refer to the citations in the reference section.

## 21.5 ACKNOWLEDGMENTS

This chapter would have not been written without the constant support and help from several colleagues and students throughout the years. We should especially thank Professor Ying Xu from Georgia University for his good advice and suggestions. We also should thank adjunct professor Yanxin Huang and students Xiumei Wang, Fangxun Sun, and Zhichao Lian for their effort and contribution to this chapter.

The chapter improved a great deal after the careful reading and the serious comments and suggestions. We are greatly indebted to Ying Sun, Juexin Wang, Zhongbo Cao, Chunbao Zhou, and Huan Yang, for their efforts to push this chapter from a draft into the final edition.

Also, all work in this chapter was supported by the NSFC (60673023, 60703025, 60803052, 10872077); the National High-Tech R&D Program of China (863) (grant 2007AA04Z114, 2009AA02Z307); the Science-Technology Development Project from Jilin Province of China (20080708, 20080172); the project of 200810026, 20091021 support by Jilin University, and "211" and "985" project of Jilin University.

## REFERENCES

1. J.Z. Zhou, D.K. Thompson, Y. Xu, and J.M. Tiedje. *Microbial Functional Genomics*. Wiley-LISS, New York, 2004.
2. X. Chen, Z.C. Su, Y. Xu, and T. Jiang. Computational prediction of operons in synechococcus sp. WH8102 [J]. *Gen Informatics*, 15(2):211–222, 2004.
3. X. Chen, Z. Su, B. Palenit, Y. Xu, and T. Jiang. Operon prediction by comparative genomics: An application to the synechococcus sp. WH8102 genome [J]. *Nucleic Acids Res*, 32(7):2147–2157, 2004.
4. R.L. Tatusov, E.V. Koonin, and D.J. Lipman. A genomic perspective on protein families. *Science*, 278:631–637, 1997.
5. M.D. Ermolaeva, O. White, S.L. Salzberg. Prediction of operons in microbial genomes. *Nucleic Acids Res*, 29:1216–1221, 2001.
6. E.J. Alm, K.H. Huang, M.N. Price, R.P. Koche, K. Keller, I.L. Dubchak, and A.P. Arkin. The microbesonline web site for comparative genomics. *Genome Res*, 15(7):1015–1022, 2005.
7. S. Okuda, T. Katayama, S. Kawashima, S. Goto, and M. Kanehisa. ODB: A database of operons accumulating known operons across multiple genomes. *Nucleic Acids Res*, 34(Special Issue):D358–D362, 2006.
8. A.M. Huerta, H. Salgado, D. Thieffry, and J. Collado-Vides, RegulonDB: A database on transcriptional regulation in Escherichia coli. *Nucleic Acids Res*, 26(1):55–59, 1998.
9. Y. Makita, M. Nakao, N. Ogasawara, and K. Nakai. DBTBS: Database of transcriptional regulation in Bacillus subtilis and its contribution to comparative genomics. *Nucleic Acids Res*, 32:D75–D77, 2004.
10. F.L. Mao, P. Dam, J. Chou, V. Olman, and Y. Xu. DOOR: A database for prokaryotic operons. *Nucleic Acids Res*, 37:D459–D463, 2009.

11. C. Sabatti, L. Rohlin, and M.K. Oh. Co-expression pattern from DNA microarray experiments as a tool for operon prediction. *Nucleic Acids Res*, 30(13):2886–2893, 2002.
12. Y. Zheng, J.D. Szustakowski, L. Fortnow, R.J. Roberts, and S. Kasif. Computational identification of operons in microbial genomes. *Genome Res*, 12:1221–1230, 2002.
13. H. Salgado, G. Moreno-Hagelsieb, T.P. Smith, and J. Collado-Vides, Operons in Escherichia coli: Genomic analyses and predictions. *Proc Natl Acad Sci*, 97(12):6652–6657, 2000.
14. M.D. Ermolaeva, O. White, S.L. Salzberg. Prediction of operons in microbial genomes. *Nucleic Acids Res*, 29(5):1216–1221, 2001.
15. M. Pellegrini, E.M. Marcotte, M.J. Thomopson, D. Eisenberg, and T.O. Yeates. Assigning protein functions by comparative genome analysis: Protein phylogenetic profiles. *Proc Natl Acad Sci*, 96(8):4285–4288, 1999.
16. D.M. Ermolaev, O. White, and S.L. Salzberg. Prediction of operons in microbial genomes[J]. *Nucleic Acids Res*, 29(5):1216–1221, 2001.
17. E. Jacob, R. Sasikumar, and K.N.R. Nair. A fuzzy guided genetic algorithm for operon prediction [J]. *Bioinformatics*, 21(8):1403–1407, 2005.
18. M. Pertea, K. Ayanbule, M. Smedinghoff, and S.L. Salzberg. OperonDB: A comprehensive database of predicted operons in microbial genomes. *Nucleic Acids Res*, 37:D479–D482, 2009.
19. I. Moszer, P. Glaser, and A. Danchin. SubtiList: A relational database for the *Bacillus subtilis* genome. *Microbiology*, 141:261–268, 1995.
20. D.L. Wheeler, C. Chappey, A.E. Lash, D.D. Leipe, T.L. Madden, G.D. Schuler, T.A. Tatusova, and B.A. Rapp. Database resources of the national center for biotechnology information. *Nucleic Acids Res*, 28:10–14, 2000.
21. P. Dam, V. Olman, K. Harris, Z. Su, and Y. Xu. Operon prediction using both genome-specific and general genome information. *Nucleic Acids Res*, 35:288–298, 2007.
22. R.W. Brouwer, O.P. Kuipers, and S.A. Hijum. The relative value of operon predictions. *Brief Bioinform*, 9(5):367–375, 2008.
23. H. Salgado, G. Moreno-Hagelsieb, T.P. Smith, and J. Collado-Vides. Operons in Escherichia coli: Genomic analyses and predictions, *Proc Natl Acad Sci*, 97(12):6652–6657, 2000.
24. M.D. Ermolaeva, O. White, and S.L. Salzberg. Prediction of operons in microbial genomes [J]. *Nucleic Acids Res*, 29(5):1216–1221, 2001.
25. M. Pellegrini, E.M. Marcotte, M.J. Thomopson, D. Eisenberg, and T.O. Yeates. Assigning protein functions by comparative genome analysis: Protein phylogenetic profiles. *Proc Natl Acad Sci*, 96(8):4285–4288, 1999.
26. D.A. Liberles, A. Thoren, G. Heijne, and A. Elofsson. The use of phylogenetic profiles of gene predictions. *Curr Genom*, 3(3):131–137, 2002.
27. J.P. Vert. A tree kernel to analyze phylogenetic profiles. *Bioinformatics*, 18(9):276–284, 2002.
28. C. Sabatti, L. Rohlin, and M.K. Oh. Co-expression pattern from DNA microarray experiments as a tool for operon prediction [J]. *Nucleic Acids Res*, 30(13):2886–2893, 2002.
29. X.M. Wang, W. Du, Y. Wang, C. Zhang, C.G. Zhou, S.Q. Wang, and Y.C. Liang. The application of support vector machine to operon prediction. *Second International Conference on Future Generation Communication and Networking, FGCN '08*, volume 3, 2008, pp. 59–62.

30. S.Q. Wang, Y. Wang, W. Du, F.X. Sun, X.M. Wang, Y.C. Liang, and C.G. Zhou. A multi-approaches-guided genetic algorithm with application to operon prediction. *Artif Intell Med*, 41:151–159, 2007.

31. W. Du, Y. Wang, S.Q. Wang, X.M. Wang, F.X. Sun, C. Zhang, C.G. Zhou, C.Q. Hu, and Y.C. Liang. Operon prediction by GRNN based on log-likelihoods and wavelet transform. *Dynam Continuous, Discrete Impulsive Syst, A Suppl, Adv Neural Networks*, 14(S1):323–327, 2007.

32. B. Tjaden, D.R. Haynor, S. Stolyar, C. Rosenow, and E. Kolker. Identifying operons and untranslated regions of transcripts using Escherichia coli RNA expression analysis. *Bioinformatics*, 18(90001):S337–S344, 2002.

33. S.M. Lin and K.F. Johnson. Methods of Microarray Data Analysis II. Kluwer Academic Publishers, Waltham, MA, 2002.

34. D.J. Spiegelhalter, N.G. Best, B.P. Carlin, and A. van der Linde. Bayesian measures of model complexity and fit (with discussion). *J Roy Stat Soc, B*, 64:583–639, 2002.

35. J. Bockhorst, M. Craven, D. Page, J. Shavlik, and J. Glasner. A Bayesian network approach to operon prediction. *Bioinformatics*, 19(10):1227–1235, 2003.

36. H. Shimizu, S. Oba, and S. Ishii. Operon prediction by DNA microarray: An approach with a bayesian network model. *Gen Informatics*, 14:310–311, 2003.

37. A. Santana and G. Provan. An analysis of bayesian network model-approximation techniques. *Proceedings of the European Conference on AI*, 2008, pp. 851–852.

38. B.P. Westover, J.D. Buhler, J.L. Sonnenburg, and S.I. Gordon. Operon prediction without a training set. *Bioinformatics*, 21(7):880–888, 2005.

39. G.Q. Zhang, Z.W. Cao, Q.M. Luo, Y.D. Cai, and Y.X. Li. Operon prediction based on SVM. *Comput Biol Chem*, 30(3):233–240, 2006.

40. C.A. Ankenbrandt. An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. *Foundations of Genetic Algorithms*, 1991, pp. 53–68.

41. M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res*, 28:27–30, 2000.

42. T.T. Tran, P. Dam, Z.C. Su, F.L. Poole, M.W. Adams, G.T. Zhou, and Y. Xu. Operon prediction in *Pyrococcus furiosus. Nucleic Acids Res*, 35(1):11–20, 2007.

# 22

# PROTEIN FUNCTION PREDICTION WITH DATA-MINING TECHNIQUES

Xing-Ming Zhao and Luonan Chen

## 22.1 INTRODUCTION

One of the most challenging problems in the postgenomic era is to annotate uncharacterized proteins with biological functions. In past decades, a huge amount of protein sequences were accumulated in public databases. However, the pace at which proteins are annotated is far behind the one at which protein sequences accumulate. Currently, about 25% of genes remain uncharacterized for the well-studied *Saccharomyces cerevisiae*, whereas only about 20% of genes are not annotated for *Homo sapiens*. It would be time consuming and expensive to determine the functions of all proteins in a lab. Computational biology that uses data mining techniques provides an alternative way to predict functions of proteins based on their sequences, structures, gene expression profiles, and so on. For instance, a straightforward way is to apply PSI-blast [1] and FASTA [63] to find homologous proteins and transfer their annotations to the target protein in which the proteins with similar sequences are assumed to carry out similar functions. However, the alignment-based methods may not work well when the sequence similarity between known proteins and the query protein is very low (*e.g.*, below 30%). Under the circumstances, the alignment-free methods provide an alternative solution to this problem by using data-mining techniques [37, 47, 90, 98], in which the alignment-free methods can detect the remote homologies of the target protein efficiently. In addition to sequences, structures have been exploited for protein annotation [18, 52] because the structures are believed to

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

be more conservative than the corresponding sequences, and proteins with similar structures are assumed to have similar functions [5].

Recently, with the advance in high-throughput biotechnologies, such as yeast two-hybrid systems [21] and microarray expression profiles [29], a large amount of biological data have been generated. These data are rich sources for deducing and understanding protein functions. Accordingly, many computational methods have been developed for protein function prediction based on these high-throughput data. For example, the protein-protein interaction (PPI) data are used for protein annotation [94] with the assumption that interacting proteins have similar functions, and gene expression profiles are used with the assumption that genes with similar expression profiles usually carry out similar functions, and so on.

To facilitate the understanding of protein functions, various databases are constructed for protein annotation with distinct definitions on protein function. For example, enzyme classification presents a nomenclature for enzymes by classifying enzymes into different groups. In the classification, each enzyme is associated with at most four numbers denoted as the enzyme commission (EC) number in which the more numbers, the more specifically the EC number describes the function of the enzyme. For instance, EC 1.1 represents “Acting on the CH-OH group of donors,” EC 1.1.1 means “With nicotinamide adenine dinucleotide (NAD) or nicotinamide adenine dinucleotide phosphate (NADP) as acceptor,” and EC 1.1.1.1 implies “alcohol dehydrogenase.” Another widely used database is Gene Ontology (GO) [3], in which each protein can be described in three categories including molecular function, biological process, and cellular component. Table 22.1 lists the popular function annotation databases that have been used widely for protein annotation in literature.

In this chapter, we aim to describe the computational methods for protein function prediction in a comprehensive manner, especially from the perspective of data-mining. In particular, we describe data-mining techniques that can predict protein functions based on various data, including protein sequences, structures, gene expression profiles, protein-protein interactions, and integration of different data. Given different biological data, we present a comprehensive framework on protein annotation based on data-mining techniques. Note that this chapter aims to summarize recent developments on protein function prediction and is by no means comprehensive because of the rapid evolvement of the field.

## 22.2 PROTEIN ANNOTATION BASED ON SEQUENCE

### 22.2.1 Protein Sequence Classification

Given an unknown protein, the most straightforward way to predict its possible functions is to align it against a reference database, such as NCBI RefSeq [68], with popular methods such as PSI-BLAST [1]. After getting the best hit for the target protein, it is reasonable to assume that the target protein has similar functions as its best hit if they have high sequence identity (*e.g.*, $\geq 90\%$). However, there is no safe sequence similarity threshold that can guarantees that two proteins have the same function.

**Table 22.1 Popular biological databases for protein annotation**

| Database | Description |
|---|---|
| COG | Clusters of orthologous groups (COG)<br>http://www.ncbi.nlm.nih.gov/COG/index.html |
| Enzyme Nomenclature | Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes by the Reactions they Catalyse<br>http://www.chem.qmul.ac.uk/iubmb/enzyme/ |
| Funcat | An annotation scheme for the functional description of proteins from prokaryotes, unicellular eukaryotes, plants and animals<br>http://mips.gsf.de/projects/funcat |
| Gene Ontology | A controlled vocabulary to describe gene and gene product attributes<br>http://www.geneontology.org/ |
| HAMAP | High-quality automated and manual annotation of microbial proteomes<br>http://www.expasy.org/sprot/hamap/ |
| KEGG | A complete computer representation of the cell, the organism,<br>http://www.genome.jp/kegg/ |
| SCOP | A detailed and comprehensive description of the structural and evolutionary relationships between all proteins whose structures are known<br>http://scop.mrc-lmb.cam.ac.uk/scop/ |
| TIGRFAMs | Protein families based on hidden Markov models and the biosphere<br>http://www.tigr.org/TIGRFAMs/ |
| Uniprot | The most comprehensive catalog of information on proteins<br>http://www.ebi.uniprot.org/index.shtml |

In particular, the sequence similarity approach fails if it is difficult to find a hit in the reference database for the target protein with high sequence identity. It has been found that it is difficult to detect the homology relationship if the sequence identity between any pair of sequences is less than 40%.

Under the circumstance, there are many data-mining methods that have been proposed to detect the homology relationship without sequence alignment, which usually is referred to as protein sequence classification. Figure 22.1 shows the flowchart of protein sequence classification using data-mining techniques. In protein sequence classification, all characterized proteins are classified into different families in advance according to their structures and functions. Accordingly, one classifier is constructed for each protein family. The unknown protein will be classified into one of the known families according to its sequence content and therefore is annotated with functions corresponding to the family. Recently, various data-mining techniques (*e.g.*, support vector machines [SVMs] [37, 47, 98] and neural networks [26]) have been applied successfully to protein sequence classification and have shown superiority to other methods.

The data-mining techniques for protein sequence classification differ in either protein representations or classifiers used. Generally, to employ data-mining techniques for protein sequence classification, each protein should be represented as a feature

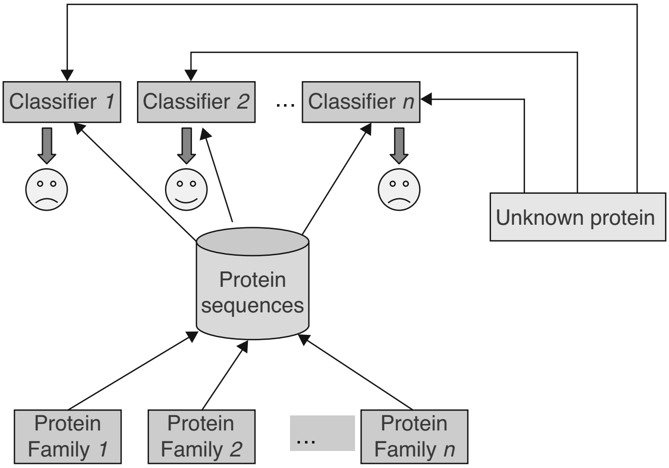


**Figure 22.1** Schematic overview of protein sequence classification.

vector. In literature, several methods have been developed for protein descriptions. For example, $k$-mer composition is one widely used description of proteins by taking into account amino acid composition and order information [93, 98]. In $k$-mer composition, a sliding window of length $k$ is moved along the protein sequence, and the frequency of the $k$-mer word is recorded in which the number of $k$-mer words is $20^k$. For example, there are 400 words for each protein sequence if a 2-mer composition is considered. The $k$-mer composition is used widely because of its simplicity and usefulness. Other methods also are developed to describe proteins based on physical and chemical properties of amino acids (*e.g.*, hydrophile and hydrophobicity [37]). Furthermore, alignment profiles taking into account evolutionary information also are used to detect homology, for example, sequence profiles [50] and the kernel method using pairwise sequence alignment [48]. Saigo *et al.* proposed new kernels for strings adapted to biological sequences, namely local alignment kernels, to detect protein homology [74]. Rangwala and Karypis proposed a new similarity between pairs of proteins using profile-to-profile scoring schemes [70]. Ben-Hur and Brutlag proposed a new method using motif composition for feature extraction [9]. Lingner and Meinicke introduced a feature vector representation for protein sequences based on distances between short oligomers and showed good performance [49]. Most recently, Damoulas and Girolami proposed a single multiclass kernel machine that informatively combines the distinct feature groups for protein homology detection [24]. Because the dimensionality of the protein vectors generated is usually very high, more computation cost may be introduced and the performance of predictor is degraded at the same time. Several techniques are presented to reduce dimensionality and noise, such as latent semantic analysis [27], SVD [69, 92], and chi-square feature selection [20].

In the data-mining approaches described, protein sequence classification is treated as a multiclass classification problem that usually is reduced to a set of binary classification problems in which one classifier is designed for each class. The proteins in one class are seen as positive examples, whereas those outside the class are seen as negative examples [47, 48]. However, the imbalance problem will result in this case because the number of proteins in one class is usually much smaller than that of proteins outside the class. As a result, the imbalanced data cause classifiers to tend to overfitt and to perform poorly on the minority class. Despite the considerable success, existing methods tend to misclassify the examples in the minority class (positive examples in this case), especially when the data are highly imbalanced. Under the circumstance, Zhao *et al.* [97] proposed a new sampling technique that uses both oversampling and undersampling to balance the data. Furthermore, an ensemble classifier was constructed for protein sequence classification in which the classifiers trained in a different feature space are combined together to improve prediction accuracy.

### 22.2.2 Protein Subcellular Localization Prediction

Generally, proteins are transported to specific compartments in a cell to function properly. Figure 22.2 shows several typical subcellular localizations in a cell. These subcellular localizations can provide insights into protein functions. Recently, various data-mining methods have been developed for protein subcellular localization prediction in which different types of protein descriptions are explored such as amino acid composition (AA) and amino acid pair composition (pAA). For instance, SubLoc [17] used SVM and sequence features to obtain high prediction accuracy. Nakashima and Nishikawa [59] used pAA, and Chou [22] used pseudoamino acid composition (PseAA) for prediction. PSORT [36] used various sequence features

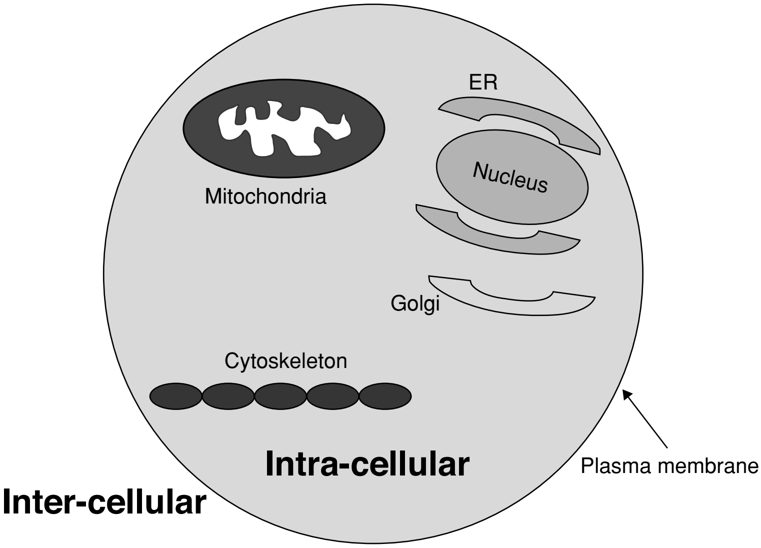


**Figure 22.2** Some typical subcellular localizations in one cell.

to predict protein localization sites in eukaryotic cells. TargetP [30] used artificial neural networks (ANN) and a N-terminal sequence to predict subcellular localizations. Furthermore, protein motifs or domains also have been useful for subcellular localization prediction. Mott *et al.* [56] used cooccurrence of domain families to predict protein cellular localization. Scott *et al.* [77] used a similar idea of motif cooccurrence and developed a Bayesian network localization predictor. Park and Kanehisa [62] used motif-like gapped amino acid composition (GapAA) and found it useful as a feature of proteins in distinct compartments.

Except for sequence composition data, other information also has been found useful for subcellular localization prediction. Cai and Chou [14] integrated different information, including pseudoamino acid composition, gene ontology information, and domain composition for subcellular localization prediction. SherLoc integrated text and protein sequence data and got high prediction accuracy. BaCelLo [66] used N-terminal, C-terminal, amino acid composition, and a sequence alignment profile to predict subcellular localizations. Tamura and Akutsu [83] used alignment of block sequences to improve prediction accuracy. Despite the success of different methods, it is difficult to say which is better. Liu *et al.* [51] presented a metapredictor by combining strengths of multiple available predicting programs, thereby obtaining high prediction accuracy. Chang *et al.* [16] developed a probabilistic latent semantic analysis method for gram-negative bacteria. Lee *et al.* [46] integrated various information including protein interaction network, gene ontology, hydrophobicity, side-chain mass and domain composition, and improved prediction accuracy significantly. However, most methods described do not consider the imbalance problem in protein subcellular localization prediction in which proteins of target compartment generally are regarded as positive samples, whereas those outside are regarded as negative samples, and the number of negative samples is usually much larger than the one of positive samples. Furthermore, distinct features contribute differently to the prediction, and therefore, feature selection may improve prediction accuracy. Most recently, Zhao and Chen developed a balanced and ensemble classifier that can improve subcellular localization prediction significantly [100]. Later, the approach was applied successfully to predict protein subcellular localizations for pathogen fungus *Fusarium graminearum* [101].

## 22.3 PROTEIN ANNOTATION BASED ON PROTEIN STRUCTURE

Although protein sequence can help to predict functions of unknown proteins, it has been found that protein structure is more conservative than protein sequence and is therefore helpful for predicting protein function. Today, some public databases deposit protein structures, such as Protein Data Bank (PDB) [10] and Structural Classification of Proteins (SCOP) [53]. Similar to sequence-based methods, one can infer the function of an unknown protein by comparing its structure with those of known proteins. In general, two proteins share functions if they have similar structures. Several structural alignment methods have been developed, such as DALI [35], CATHEDRAL [72], and SAMO [18]. Despite the success of structure in predicting protein function, it has been found that two proteins may have distinct functions even though they have very similar structures [6]. On the other hand, two proteins may perform the same function even though they have very different structures [91]. In other words, it is never safe to conclude that two proteins have the same function if they have similar structures.

Because proteins function by interacting with each other, protein surface area through which proteins interact provides insights into the relationship between protein structure and function [39]. Therefore, the function of one known protein can be transferred to an uncharacterized protein, between which there is a matched surface area. In literature, several methods have been developed to analyze protein surfaces, such as CASTp [11] and eF-Site [42]. Most recently, Liu *et al.* proposed a new method to predict protein function by constructing a pocket similarity network [52] in which the pockets are surface patterns defined in the CASTp database. The numerical results demonstrate that the pocket similarity network not only is effective for identifying remote homologues but also reveals direct links between small surface patterns and GO functions.

## 22.4 PROTEIN FUNCTION PREDICTION BASED ON GENE EXPRESSION DATA

In the last decade, gene expression data generated by microarray technologies are proven useful for elucidating gene function in which genes sharing similar functions tend to coexpress. Therefore, many data-mining methods including both supervised and unsupervised approaches have been used to predict protein functions based on gene expression. Generally, gene expression data are organized as a matrix in which each gene is denoted by a vector (row) and each condition by an attribute (column). In supervised methods, the functions of some genes are known in advance and act as class labels for corresponding vectors representing genes. The other genes remain unlabeled, and the data-mining techniques are used to assign a label to them. Brown *et al.* [13] applied SVMs to predict the functions of a yeast genes based on gene expression data. The numerical experiments demonstrate the promising effectiveness of SVMs in protein function prediction. Mateos *et al.* [55] uszed multilayer perceptrons (MLP) to predict protein functions of a yeast genome under 96 function classes. By analyzing the performance of the MLP classifier, the authors showed that the performance of the classifier is affected not only by the learning technique but also by the nature of the data. Later, Ng and Tan [60] combined multiple datasets for learning with SVMs and presented a strategy to select the most informative data sets for learning individual classes. Recently, Pandey and Kumar [61] presented a modified $k$-nearest neighbor learning algorithm for protein annotation based on gene expression data in which the similarity between functional classes is taken into account. The results demonstrate that the incorporation of interrelationships between functional classes substantially improves the performance of function prediction algorithms.

In unsupervised methods, it is assumed that no label is known in advance. The unsupervised techniques, especially clustering methods, group genes that coexpress in various conditions with the idea that genes with similar expression profiles share common regulatory mechanisms and functions. For example, Raychaudhuri *et al.* used principal component analysis to identify gene expression signatures from gene expression data [71]. Tavazoie *et al.* used $k$-means approach to cluster gene expression data [85]. Other clustering methods, such as hierarchical clustering [29] and self-organizing maps [82], also are used widely to predict gene function by clustering gene expression data. Recently, biclustering methods also are used to find patterns of coregulated genes in which coregulated genes are assumed to share similar functions [8, 67]. Unlike the clustering methods described, biclustering techniques discover coregulated genes not only over genes but also over conditions (samples). Therefore, this technique can discover patterns ignored by other methods.

## 22.5 PROTEIN FUNCTION PREDICTION BASED ON PROTEIN INTERACTOME MAP

In a cell, proteins function by interacting with each other, and PPIs thereby provide an alternative way to protein function prediction. Therefore, it is reasonable to assume that interacting proteins share common functions (*i.e.*, "Guilt by Association" rule). Several methods have been proposed to predict protein functions based on protein interactome. These approaches can be grouped into two categories (*i.e.*, function prediction based on local and global topology structures of interaction map respectively, which will be addressed in detail below).

### 22.5.1 Protein Function Prediction Based on Local Topology Structure of Interaction Map

A straightforward way to predict the function of an unknown protein using protein interaction is to transfer the function of its direct interaction partners that have been annotated to the target protein (see Figure 22.3a). Schwikowski *et al.* [76] annotated an unknown protein with the functions occurring most often for its interaction partners, which is called the majority rule method in literature. Unfortunately, the majority rule method will not work if there are no annotations for the direct interaction partners of the target protein (see Figure 22.3b). To handle this problem, Hishigaki *et al.* [34] defined the neighborhood of a protein with a radius of $r$. For an unknown protein, the functional enrichment in its $r$-neighborhood is investigated with a $\chi^2$ test, and the top ranking functions are assigned to the unknown proteins (see Figure 22.3b). This approach alleviates the limitations of the majority rule method to some extent.

In addition to the neighborhood of the protein of interest, the shared neighborhood of a pair of proteins also are considered recently. Chua *et al.* [23] defined the functional similarity between a pair of proteins by considering the direct and indirect neighbors of the target protein pair. Samanta and Liang [75] defined the

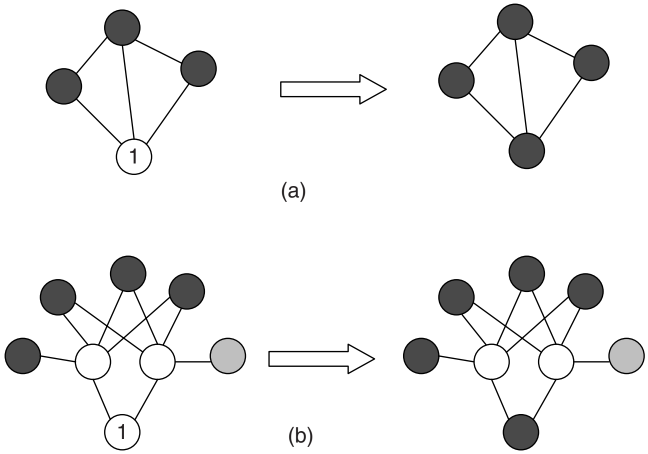


**Figure 22.3** Predicting the function of protein 1 based on protein-protein interactions in which different colors denote different functions, and nodes without any color denote uncharacterized proteins. (a) Annotating protein 1 based on the information of its direct interaction partners, where protein 1 is annotated with a function represented by blue; (b) Annotating protein 1 based on its interaction neighborhood information, and protein 1 is annotated with a function represented by blue.

probability of two proteins sharing common functions with a hypergeometric model. Deng *et al.* [25] developed a Markov random field (MRF) model for protein function prediction based on information from both interaction and noninteraction partners. Kirac *et al.* [43] presented a model that considers the annotations in the paths leading to the target protein in the PPI network. This model is implemented using the probabilistic suffix tree data structure, and the results are better than other neighborhood-based methods.

The methods described are actually generative models that construct a model with only positive samples (*i.e.*, annotated proteins) whereas the proteins outside of the target functional class usually are seen as negative samples. However, this may be not true because each protein usually is annotated with multiple functions. Although some proteins are not annotated with the target function currently, they actually may have the function. Furthermore, the imbalanced problem will develop if all proteins outside of the functional class are seen as negative samples, which will degrade the performance of the classifier [97]. Recently, Zhao *et al.* [96] proposed a new algorithm to define the negative samples in protein function prediction. In detail, the one-class SVMs and two-class SVMs are used as the core learning algorithm to find the representative negative samples so that the positive samples hidden in the unlabeled data can be best recovered. The experiments demonstrate that with the negative samples generated, the performance of prediction methods is improved compared with other methods defining negative samples [15].

In addition, some clustering methods are developed to cluster the protein interactome according to its topology and identify some modules including protein complex in which proteins belonging to the same module generally share common function. Bader and Hogue [4] proposed the molecular complex detection algorithm (MCODE) to predict complexes in a PPI network. Pereira-Leal *et al.* [65] applied the Markov clustering (MCL) [31] algorithm to predict complexes in the PPI network. The authors showed that pathways also can be inferred from the hierarchical network of modular interactions. King *et al.* [41] proposed the restricted neighborhood search clustering (RNSC) algorithm to partition the PPI network into clusters based on a cost function that is used to evaluate the partitioning. The authors showed that the RNSC algorithm outperforms the MCODE algorithm. Later, Dunn *et al.* [28] used the edge-betweenness clustering to predict complexes in a PPI network. Recently, a study [12] compared different clustering methods for complex prediction in the PPI network. The authors compared four methods: RNSC, SPC [81], MCODE, and MCL. With a set of known complexes in the PPI network, they examined the performance of the four methods in detecting complexes in perturbed PPI networks. The authors showed that the MCL algorithm is more robust compared with others.

### 22.5.2 Protein Function Prediction Based on Global Topology of Interaction Map

The methods described explore only local topology structure of protein interaction map, which may not work if there are no annotations for all neighbors of the target protein (*e.g.*, Figure 22.3b). On the other hand, the global topology information may improve prediction accuracy. Zhou *et al.* [103] predicted protein function using the shortest path distance between a pair of genes based on gene expression data. Later, Arnau *et al.* [2] used the shortest path length among proteins as a similarity measure for hierarchical clustering. Vazquez *et al.* [89] proposed a new global method to annotate a protein considering the topology of the whole network. A function is assigned to an unknown protein so that the number of the same annotations associating with its neighbors is maximized by minimizing the score function:

$$E = -\sum_{i,j} J_{ij}\delta(\sigma_i, \sigma_j) - \sum_{i} h_i(\sigma_i) \tag{22.1}$$

where $J_{ij}$ is the element of the adjacency matrix of the interaction network, $\delta(\sigma_i, \sigma_j)$ is the discrete $\delta$ function, and $h_i(\sigma_i)$ is the number of partners of protein $i$ annotated with function $\sigma_i$. The simulated annealing is employed to minimize the score function. Later, Karaoz *et al.* [40] developed a similar method by assigning a state $s_u \in \{0, 1\}$ to an unknown protein $u$ to maximize the score function $\sum_{(u,v)\in E} s_u s_v$, where $u$ and $v$ are nodes in the PPI network, and $(u, v) \in E$ means there is an interaction between protein $u$ and protein $v$. The optimization problem is handled by employing a discrete Hopfield network in which only one function is considered each time.

Recently, another method proposed by Nabieva *et al.* [58] also formulates the annotation problem as a global optimization problem in which a unique function is assigned to an unknown protein to minimize the cost of edges connecting proteins with different assignments. In detail, they formulated the optimization problem as an integer linear program (ILP) model. In the ILP model, each protein annotated with the target function in the PPI network is regarded as the source of functional flow. By simulating the spread of this functional flow through the network, each unknown protein gets a score for having the function based on the amount of flow that it received.

## 22.6 PROTEIN FUNCTION PREDICTION BASED ON DATA INTEGRATION

Although various kinds of high-throughput data can give hints about protein functions, many high-throughput data are notorious for the noise in the data and the specificity for scale. Recently, integration of different types of biological data for protein function prediction is becoming a popular trend and is expected to improve prediction accuracy. There are many ways to comb different kinds of data sources for protein annotation, including the Bayesian network, kernel methods, and so on.

Troyanskaya *et al.* [87] developed a Multisource Association of Genes by Integration of Clusters (MAGIC) system in which the Bayesian network is employed to integrate different types of high-throughput biological data. The inputs of the system are gene-gene relationship matrices established on different high-throughput data. The numerical results demonstrate that MAGIC improves prediction accuracy compared with microarray analysis alone. Chen and Xu [19] also developed a Bayesian model to integrate different kinds of data sources including protein-protein interaction, microarray data, and protein complex data. In their methods, two prediction models are presented (*i.e.*, local prediction and global prediction). In the local prediction model, the probability of protein $x$ having function $F$ is defined as:

$$G(F, x) = 1 - \prod_{i=1}^{m}(1 - P(S|D_i)) \tag{22.2}$$

where $m$ is the total number of high-throughput data sources, and $P(S|D_i)$ is the probability that two genes have the same function given data $D_i$, which is defined as follows:

$$P(S|D_i) = 1 - \prod_{j=1}^{n}(1 - P_j(S|D_i)) \tag{22.3}$$

where $n$ is the total number of interaction partners given $D_i$, and $P_j(S|D_i)$ is the probability that the interacting pair $j$ have the same function. In the local prediction method, only immediate interaction partners are considered and thereby may lead to local optimal solutions. Therefore, a global prediction model is presented by using the Boltzmann machine to characterize the global stochastic behavior of the network. The authors showed that the global model outperforms other existing methods.

Lanckriet *et al.* [44] developed a kernel method for data fusion. They first constructed a kernel matrix $K_i$ for each data source $i$ and then combined all kernel matrices in the following linear form:

$$K = \sum_{i=1}^{m} \mu_i K_i \tag{22.4}$$

where $m$ is the total number of kernel matrices. The coefficients $\mu_i$ are estimated via a semidefinite program (SDP). The authors showed that the kernel fusion method outperforms the MRF method [25]. Recently, Zhao *et al.* [95] constructed a functional linkage graph from different types of data with the shortest path distance as the similarity measure and then employed SVMs to annotate proteins. Tsuda *et al.* [88] proposed a kernel method by combining multiple protein networks in which the combination weights are obtained by convex optimization. Barutcuoglu *et al.* [7] provided another way to integrate different types of data for protein function prediction. They combined different types of data into one vector by concatenating all feature vectors for one gene. With the data available, they trained a SVM classifier for each functional class. Furthermore, a Bayesian net is constructed to combine the outputs of the classifiers considering the functional taxonomy.

Hanisch *et al.* [33] constructed a distance function by combining information from gene expression data and biological networks. Based on the distance function, a joint clustering of genes and vertices of the network is performed. Segal *et al.* [78] described a probabilistic model that is learned from the data using the expectation maximization (EM) algorithm to detect pathways from gene expression and protein interaction data. Tornow and Mewes [86] calculated the correlation strength of a group of genes considering the probability of these genes belonging to the same module in a different network. The rational behind the method is that the group of genes with a significant correlation strength in different networks have a high probability that they perform the same function. Tanay *et al.* [84] presented an integrative framework Statistical-Algorithmic Method for Bicluster Analysis (SAMBA) for various kinds of data including protein interaction, gene expression, phenotypic sensitivity, and transcription factor (TF) binding. The authors proved the effectiveness of SAMBA by predicting the functions of >800 uncharacterized genes. Later, Massjouni *et al.* [54] constructed a functional linkage network (FLN) from gene expression and molecular interaction data and propagated the functional labels across the FLN to predict precisely the functions of unlabelled genes. Shiga *et al.* [80] proposed a hidden modular random field model for protein annotation by combining gene expression data and gene network.

Generally, proteins in the same pathway share a common function. Therefore, it is possible to annotate proteins by identifying components that belong to the same pathway. Most recently, Zhao *et al.* proposed a novel integer linear programming

(ILP) model to predict signaling pathways by integrating PPI and gene expression data. The ILP model for uncovering a signaling pathway is described as follows:

$$\text{Minmize}_{\{x_i, y_{ij}\}} \qquad S = -\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} w_{ij} y_{ij} + \lambda \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} y_{ij} \tag{22.5}$$

$$\text{Subject to} \quad y_{ij} \le x_i, \tag{22.6}$$

$$y_{ij} \le x_j, \tag{22.7}$$

$$\sum_{j=1}^{|V|} y_{ij} \ge 1, \ \text{ if } i \text{ is either a starting or ending protein,} \tag{22.8}$$

$$\sum_{j=1}^{|V|} y_{ij} \ge 2x_i, \ \text{ if } i \text{ is not a starting or ending protein,} \tag{22.9}$$

$$x_i = 1\ , \ \text{ if } i \text{ is a protein known in signaling pathway,} \tag{22.10}$$

$$x_i \in \{0, 1\}, \ i = 1, 2, \cdots, |V|, \tag{22.11}$$

$$y_{ij} \in \{0, 1\}, \ i, j = 1, 2, \cdots, |V|, \tag{22.12}$$

where $w_{ij}$ is the weight of edge $E(i, j)$ in the undirected weighted network $G$, $x_i$ is a binary variable for protein $i$ to denote whether protein $i$ is selected as a component of the signaling pathway, and $y_{ij}$ is also a binary variable to denote whether the biochemical reaction represented by $E(i, j)$ is a part of the signaling pathway. $\lambda$ is a positive penalty parameter to control the trade-off between the signaling pathway weight and size, and $|V|$ is the total number of proteins in the PPI network. The constraint $\sum_j^{|V|} y_{ij} \ge 2x_i$ is to ensure that $x_i$ has at least two linking edges once it is selected as a component of the signaling pathway, whereas the constraint $\sum_j^{|V|} y_{ij} \ge 1$ means that each starting protein or ending protein has at least one link to or from other proteins. These two constraints ensure that the components in the subnetwork are as connected as possible. The constraints $y_{ij} \le x_i$ and $y_{ij} \le x_j$ mean that if and only if proteins $i$ and $j$ are selected as the components of signaling pathway, then the biochemical then reaction denoted by the edge $E(i, j)$ should be considered. Equation (22.22.10) is the condition for any protein known involved in the signaling pathway (*e.g.*, from the experiment results or literature). The results on yeast MAPK signaling pathways demonstrate the predictive power of the ILP model.

## 22.7 CONCLUSIONS AND PERSPECTIVES

Protein function prediction has been an active field in computational biology in the past years. Despite the numerous computational methods that have been developed, it is still a difficult task to annotate uncharacterized proteins precisely. It is not trivial to say which method is better than others while predicting protein functions, and it is usually difficult for the newcomers to determine which method should be used to predict the functions of unknown proteins. Therefore, a systematic comparison of different methods is needed. By comparing the prediction methods with golden standard dataset (*e.g.*, GO [3] and MIPS Funcat [73]) it may provide advice to the newcomer about which methods should be used for a specific dataset. For instance, it has been shown [79] that the supervised method (*e.g.*, majority rule) [76] generally performs better than the unsupervised method (*e.g.*, MCODE) [4]. However, choosing an appropriate method involves many factors and should be conducted carefully. Generally, the supervised methods perform best if there are sufficient training data available. The unsupervised methods generally are regarded as the final choice. On the other hand, different prediction methods have been proposed for different data and functional schemas. For example, some methods perform well in one case but badly in another case. One possible solution is to construct metaservers by combining a set of best-performing prediction methods. Recently, several groups, from which popular protein prediction methods come, participated in a competition of predicting the function of *Mus musculus* proteins [64]. Diverse and independently developed computational approaches for predicting protein function are compared on the same real data. Surprisingly, no method always can perform better than the others through the competition. Some methods perform best when the number of samples is limited, whereas some perform best for certain functions. The performance of these methods make it clear that different methods can complement each other, and one should try different approaches while predicting the function of a new protein. Although, numerous computation methods are available for protein annotation.

Currently, most existing computation methods consider one function each time. In other words, the biological functions are treated independently. As pointed out recently by several works [45, 61, 7], the biological functions are not independent of each other, and the incorporation of correlations among functions can improve significantly the performance of prediction methods. Therefore, the correlation among functions should be taken into account by the methods developed in the future. Another possible alternative to annotate uncharacterized proteins is to explore their domain contents. Protein domains are structural and functional units of proteins, and thereby the functions of domains can provide insights into protein functions. There are some public databases for proteins domains available, such as Pfam [32] and InterPro [38]. Recently, Zhao *et al.* presented a framework for predicting the functions of protein domains [99]. With these annotated domains, it becomes easy to annotate corresponding proteins in which these domains are located.

Generally, most prediction methods described focus on single data sources (*e.g.*, PPI or gene expression data). However, the PPI data is notorious for false positives and incompleteness. One possible solutions is to reconstruct the PPI network by removing spurious edges and adding biologically valid ones. On the other hand, the correlation coefficients among genes generally are used as the similarity measures for proteins pairs in which genes that have high correlations are assumed to have similar functions. The problem is that the correlation coefficients cannot capture the functional linkages among proteins in some cases. For example, the correlations among all proteins are very high or very low. One possible solutions is to used the higher order statistics to capture the functional linkages as suggested in [102]. Although the integration of different kinds of data has been used for protein annotation and has shown promising results, the diverse kinds of data sources should be combined carefully. Which kinds of data should be combined? Does the integration of all kinds of data really improve prediction accuracy? As mentioned in [57], different data have their own specific structure. In some cases, the combination of different kinds of data adds noise to existing data and degrade the performance of the prediction method. Therefore, the nature of the data should be taken into account while developing new prediction methods. On the other hand, new effective data fusion methods are needed in the future for protein annotation.

Despite the limitations of existing computational methods and the noise lying in the high-throughput data, protein annotation is a promising and active research field. With the rapid advance in biotechnology, more biological data with high quality and reliability will be expected. Accordingly, the prediction accuracy of function prediction methods will be improved. Although many computational methods have been developed, there is still much room for improvement. The scientists in the data-mining field are expected to develop more reliable prediction methods and to make them accessible to the biologists.

## REFERENCES

1. S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–3402, 1997.
2. V. Arnau, S. Mars, and I. Marn. Iterative cluster analysis of protein interaction data. *Bioinformatics*, 21:364–378, 2005.
3. M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, *et al.* Gene ontology: Tool for the unification of biology the gene ontology consortium. *Nat Genet*, 25:25–29, 2000.
4. G. Bader and C. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.
5. D. Bandyopadhyay, J. Huan, J. Liu, J. Prins, J. Snoeyink, W. Wang, and A. Tropsha. Structure-based function inference using protein family-specific fingerprints. *Protein Sci*, 15:1537–1543, 2006.
6. G.J. Bartlett, N. Borkakoti, and J.M. Thornton. Catalysing new reactions during evolution: Economy of residues and mechanism. *J Mol Biol*, 331:829–860, 2003.
7. Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22:830–836, 2006.
8. A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene-expression data: The order-preserving submatrix problem. *J Comput Biol*, 10:373–384, 2003.
9. A. Ben-Hur and D. Brutlag. Remote homology detection: A motif based approach. *Bioinformatics*, 19:i26–33, 2003.

10. H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28:235–242, 2000.
11. T. Binkowski, S. Naghibzadeh, and J. Liang. Castp: Computed atlas of surface topography of proteins. *Nucleic Acids Res*, 31:3352–3355, 2003.
12. S. Brohee and J. van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7:488, 2006.
13. M.P.S. Brown, W.N. Grundy, D. Lin, N. Cristianini, C.W. Sugnet, T.S. Furey, J.M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene-expression data by using support vector machines. *Proc Natl Acad Sci U S A*, 97:262–267, 2000.
14. Y. Cai and K. Chou. Predicting subcellular localization of proteins in a hybridization space. *Bioinformatics*, 20:1151–1156, 2004.
15. R.J. Carter, I. Dubchak and S.R. Holbrook. A computational approach to identify genes for functional RNAs in genomic sequences. *Nucleic Acids Res*, 29:3928–3938, 2001.
16. J.M.M. Chang, E.C.Y.C. Su, A. Lo, H.S.S. Chiu, T.Y.Y. Sung, and W.L.L. Hsu. Psldoc: Protein subcellular localization prediction based on gapped-dipeptides and probabilistic latent semantic analysis. *Proteins*, 72(2):693–710, 2008.
17. H. Chen, N. Huang, and Z. Sun. SubLoc: A server/client suite for protein subcellular location based on SOAP. *Bioinformatics*, 22:376–377, 2006.
18. L. Chen, L.Y. Wu, Y. Wang, S. Zhang, X.S. Zhang. Revealing divergent evolution, identifying circular permutations and detecting active-sites by protein structure comparison. *BMC Struct Biol*, 6:18, 2006.
19. Y. Chen and D. Xu. Global protein function annotation through mining genome-scale data in yeast Saccharomyces cerevisiae. *Nucleic Acids Res*, 32:6414–6424, 2004.
20. B.Y.M. Cheng, J.G. Carbonell, and J. Klein-Seetharaman. Protein classification based on text document classification techniques. *Proteins*, 58:955–970, 2005.
21. C. Chien, P. Bartel, R. Sternglanz, and S. Fields. The two-hybrid system: A method to identify and clone genes for proteins that interact with a protein of interest. *Proc Natl Acad Sci U S A*, 88:9578–9582, 1991.
22. K. Chou. Prediction of protein cellular attributes using pseudoamino acid composition. *Proteins*, 43:246–255, 2001.
23. H.N. Chua, W.K. Sung, and L. Wong. Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. *Bioinformatics*, 22:1623–1630, 2006.
24. T. Damoulas and M.A. Girolami. Probabilistic multiclass multi-kernel learning: On protein fold recognition and remote homology detection. *Bioinformatics*, 24:1264–1270, 2008.
25. M. Deng, K. Zhang, S. Mehta, T. Chen, and F. Sun. Prediction of protein function using protein-protein interaction data. *J Comput Biol*, 10:947–960, 2003.
26. C.H. Ding and I. Dubchak. Multiclass protein fold recognition using support vector machines and neural networks. *Bioinformatics*, 17:349–358, 2001.
27. Q.W. Dong, X.L. Wang, and L. Lin. Application of latent semantic analysis to protein remote homology detection. *Bioinformatics*, 22:285–290, 2006.
28. R. Dunn, F. Dudbridge, and C. Sanderson. The use of edge-betweenness clustering to investigate biological function in protein interaction networks. *BMC Bioinformatics*, 6:39, 2005.

29. M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc Natl Acad Sci U S A*, 95:14863–14868, 1998.

30. O. Emanuelsson, H. Nielsen, S. Brunak, and von G. Heijne. Predicting subcellular localization of proteins based on their n-terminal amino acid sequence. *J Mol Biol*, 300:1005–1016, 2000.

31. A.J. Enright, C.A. Van Dongen, and Q. Sand. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res*, 30:1575–1584, 2002.

32. R.D. Finn, J. Tate, J. Mistry, P.C. Coggill, S.J. Sammut, H.R. Hotz, G. Ceric, K. Forslund, S.R. Eddy, E.L.L. Sonnhammer, and A. Bateman. The Pfam protein families database. *Nucleic Acids Res*, 36:D281–D288, 2008.

33. D. Hanisch, A. Zien, R. Zimmer, and T. Lengauer. Co-clustering of biological networks and gene-expression data. *Bioinformatics*, 18:S145–S154, 2002.

34. H. Hishigaki, K. Nakai, T. Ono, A. Tanigami, and T. Takagi. Assessment of prediction accuracy of protein function from protein–protein interaction data. *Yeast*, 18:523–531, 2001.

35. L. Holm and C. Sander. Dali/FSSP classification of three-dimensional protein folds. *Nucleic Acids Res*, 25:231–234, 1997.

36. P. Horton, K. Park, T. Obayashi, N. Fujita, H. Harada, C. Adams-Collier, and K. Nakai. Wolf psort: Protein localization predictor. *Nucleic Acids Res*, 35:D314–D316, 2007.

37. D.S. Huang, X.M. Zhao, G.B. Huang, and Y.M. Cheung. Classifying protein sequences using hydropathy blocks. *Pattern Recogn*, 39:2293–2300, 2006.

38. S. Hunter, R. Apweiler, T.K. Attwood, A. Bairoch, A. Bateman, D. Binns, P. Bork, U. Das, L. Daugherty, L. Duquenne *et al.* InterPro: The integrative protein signature database. *Nucleic Acids Res*, 37:D211–215, 2009.

39. S. Jones and J. Thornton. Analysis of protein-protein interaction sites using surface patches. *J Mol Biol*, 272:121–132, 1997.

40. U. Karaoz, T.M. Murali, S. Letovsky, Y. Zheng, C. Ding, C.R. Cantor, and S. Kasif. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proc Natl Acad Sci U S A*, 101:2888–2893, 2004.

41. A.D. King, N. Przulj, and I. Jurisica. Protein complex prediction via cost-based clustering. *Bioinformatics*, 20:3013–3020, 2004.

42. K. Kinoshita, J. Furui, and H. Nakamura. Identification of protein functions from a molecular surface database, ef-site. *J Struct Funct Genomics*, 2:9–22, 2002.

43. M. Kirac, G. Ozsoyoglu, and J. Yang. Annotating proteins by mining protein interaction networks. *Bioinformatics*, 22:e260–e270, 2006.

44. G.R. Lanckriet, M. Deng, N. Cristianini, M.I. Jordan, and W.S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. Biocomputing 2004, Proceedings of the Pacific Symposium Hawaii, USA. ISBN 981-238-598-3, pp. 300–311.

45. H. Lee, Z. Tu, M. Deng, F. Sun, and T. Chen. Diffusion kernel-based logistic regression models for protein function prediction. *OMICS: J Int Biol*, 10:40–55, 2006.

46. K. Lee, H.Y. Chuang, A. Beyer, M.K. Sung, W.K. Huh, B. Lee, and T. Ideker. Protein networks markedly improve prediction of subcellular localization in multiple eukaryotic species. *Nucleic Acids Res*, 36:e136, 2008.

47. C.S. Leslie, E. Eskin, A. Cohen, J. Weston, and W.S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20:467–476, 2004.

48. L. Liao and W.S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. *RECOMB '02: Proceedings of the Sixth Annual International Conference on Computational Biology*. ACM, New York, 2002, pp. 225–232.

49. T. Lingner and P. Meinicke. Remote homology detection based on oligomer distances. *Bioinformatics*, 22:2224–2231, 2006.

50. B. Liu, X. Wang, L. Lin, Q. Dong, and X. Wang. A discriminative method for protein remote homology detection and fold recognition combining top-n-grams and latent semantic analysis. *BMC Bioinformatics*, 9:510, 2008.

51. J. Liu, S. Kang, C. Tang, L.B. Ellis, and T. Li. Meta-prediction of protein subcellular localization with reduced voting. *Nucleic Acids Res*, 35:e96, 2007.

52. Z. Liu, L.Y. Wu, Y. Wang, X.S. Zhang, and L. Chen. Predicting gene ontology functions from protein's regional surface structures. *BMC Bioinformatics*, 8:475, 2007.

53. L. Lo Conte, B. Ailey, T.J.P. Hubbard, S.E. Brenner, A.G. Murzin, and C. Chothia. SCOP: A structural classification of proteins database. *Nucleic Acids Res*, 28:257–259, 2000.

54. N. Massjouni, C.G. Rivera, and T.M. Murali. VIRGO: Computational prediction of gene functions. *Nucleic Acids Res*, 34:W340–W344, 2006.

55. A. Mateos, J. Dopazo, R. Jansen, Y. Tu, M. Gerstein, and G. Stolovitzky. Systematic learning of gene functional classes from DNA array expression data by using multilayer perceptrons. *Genome Res*, 12:1703–1715, 2002.

56. R. Mott, J. Schultz, P. Bork, and C.P. Ponting. Predicting protein cellular localization using a domain projection method. *Genome Res*, 12:1168–1174, 2002.

57. C.L. Myers and O.G. Troyanskaya. Context-sensitive data integration and prediction of biological networks. *Bioinformatics*, 23:2322–2330, 2007.

58. E. Nabieva, K. Jim, A. Agarwal, B. Chazelle, and M. Singh. Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 21:i302–310, 2005.

59. H. Nakashima and K. Nishikawa. Discrimination of intracellular and extracellular proteins using amino acid composition and residue-pair frequencies. *J Mol Biol*, 238:54–61, 1994.

60. S.K. Ng and S.H. Tan. On combining multiple microarray studies for improved functional classification by whole-dataset feature selection. *Genome Informatics*, 14:44–53, 2003.

61. G. Pandey and V. Kumar. Incorporating functional inter-relationships into algorithms for protein function prediction. *ISMB Satellite Meeting on Automated Function Prediction*, 2007.

62. K. Park and M. Kanehisa. Prediction of protein subcellular locations by support vector machines using compositions of amino acids and amino acid pairs. *Bioinformatics*, 19:1656–1663, 2003.

63. W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85:2444–2448, 1988.

64. L. Pena-Castillo, M. Tasan, C. Myers, H. Lee, T. Joshi, C. Zhang, Y. Guan, M. Leone, A. Pagnani, W. Kim, *et al.* A critical assessment of mus musculus gene function prediction using integrated genomic evidence. *Genome Biol*, 9:S2, 2008.

65. J.B. Pereira-Leal, A.J. Enright, and C.A. Ouzounis. Detection of functional modules from protein interaction networks. *Proteins*, 54:49–57, 2004.

66. A. Pierleoni, P.L. Martelli, P. Fariselli, and R. Casadio. BaCelLo: A balanced subcellular localization predictor. *Bioinformatics*, 22:e408–416, 2006.

67. A. Prelic, S. Bleuler, P. Zimmermann, A. Wille, P. Buhlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. A systematic comparison and evaluation of biclustering methods for gene-expression data. *Bioinformatics*, 22:1122–1129, 2006.

68. K.D. Pruitt, T. Tatusova, and D.R. Maglott. NCBI Reference Sequence (RefSeq): A curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Res*, 33:D501–504, 2005.

69. R. Rakotomalala and F. Mhamdi. Combining feature selection and feature reduction for protein classification. *SMO'06: Proceedings of the 6th WSEAS International Conference on Simulation, Modelling and Optimization*. World Scientific and Engineering Academy and Society, Stevens Point, WI, 2006, pp. 444–451.

70. H. Rangwala and G. Karypis. Profile-based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21:4239–4247, 2005.

71. S. Raychaudhuri, J.M. Stuart, R.B. Altman. Principal components analysis to summarize microarray experiments: application to sporulation time series. *Pacific Symposium on Biocomputing Pacific Symposium on Biocomputing*, 2000, pp. 455–466.

72. O.C. Redfern, A. Harrison, T. Dallman, F.M. Pearl, and C.A. Orengo. Cathedral: A fast and effective algorithm to predict folds and domain boundaries from multidomain protein structures. *PLoS Comput Biol*, 3:e232, 2007.

73. A. Ruepp, A. Zollner, D. Maier, K. Albermann, J. Hani, M. Mokrejs, I. Tetko, U. Guldener, G. Mannhaupt, M. Munsterkotter, *et al.* The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic Acids Res*, 32:5539–5545, 2004.

74. H. Saigo, J.P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689, 2004.

75. M.P. Samanta and S. Liang. Predicting protein functions from redundancies in large-scale protein interaction networks. *Proc Natl Acad Sci U S A*, 100:12579–12583, 2003.

76. B. Schwikowski, P. Uetz, and S. Fields. A network of protein-protein interactions in yeast. *Nat Biotechnol*, 18:1257–1261, 2000.

77. M.S. Scott, D.Y. Thomas, and M.T. Hallett. Predicting subcellular localization via protein motif co-occurrence. *Genome Res*, 14:1957–1966, 2004.

78. E. Segal, H. Wang, and D. Koller. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics*, 19:i264–i272, 2003.

79. R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Mol Syst Biol*, 3:88, 2007.

80. M. Shiga, I. Takigawa, and H. Mamitsuka. Annotating gene function by combining expression data with a modular gene network. *Bioinformatics*, 23:i468–i478, 2007.

81. V. Spirin and L.A. Mirny. Protein complexes and functional modules in molecular networks. *Proc Natl Acad Sci U S A*, 100:12123–12128, 2003.

82. P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E.S. Lander, and T.R. Golub. Interpreting patterns of gene-expression with self-organizing maps:

Methods and application to hematopoietic differentiation. *Proc Natl Acad Sci U S A*, 96:2907–2912, 1999.

83. T. Tamura and T. Akutsu. Subcellular location prediction of proteins using support vector machines with alignment of block sequences using amino acid composition. *BMC Bioinformatics*, 8:466, 2007.

84. A. Tanay, R. Sharan, M. Kupiec, and R. Shamir. Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proc Natl Acad Sci U S A*, 101:2981–2986, 2004.

85. S. Tavazoie, J.D. Hughes, M.J. Campbell, R.J. Cho, and G.M. Church. Systematic determination of genetic network architecture. *Nat Genet*, 22:281–285, 1999.

86. S. Tornow and H.W. Mewes. Functional modules by relating protein interaction networks and gene expression. *Nucleic Acids Res*, 31:6283–6289, 2003.

87. O.G. Troyanskaya, K. Dolinski, A.B. Owen, R.B. Altman, and D. Botstein. A Bayesian framework for combining heterogeneous data sources for gene function prediction (in Saccharomyces cerevisiae). *Proc Natl Acad Sci U S A*, 100:8348–8353, 2003.

88. K. Tsuda, H. Shin and B. Scholkopf. Fast protein classification with multiple networks. *Bioinformatics*, 21:ii59–ii65, 2005.

89. A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction from protein-protein interaction networks. *Nat Biotechnol*, 21:697–700, 2003.

90. S. Vinga and J. Almeida. Alignment-free sequence comparison-a review. *Bioinformatics*, 19:513–523, 2003.

91. J.C. Whisstock and A.M. Lesk. Prediction of protein function from protein sequence and structure. *Q Rev Biophys*, 36:307–340, 2003.

92. C. Wu, S. Shivakumar, H.P. Lin, S. Veldurti, and Y. Bhatikar. Neural networks for molecular sequence classification. *Math Comput Simul*, 40:23–33, 1995.

93. C.H. Wu, G. Whitson, J. McLarty, A. Ermongkonchai, and T. Chang. Protein classification artificial neural system. *Protein Sci*, 1:667–677, 1992.

94. S. Zhang, G. Jin, X.S. Zhang, and L. Chen. Discovering functions and revealing mechanisms at molecular level from biological networks. *Proteomics*, 7:2856–2869, 2007.

95. X.M. Zhao, L. Chen and K. Aihara. Gene function prediction with the shortest path in functional linkage graph. *Lect Notes Oper Res*, 7:68–74, 2007.

96. X.M. Zhao, L. Chen, and K. Aihara. Gene function prediction using labeled and unlabeled data. *BMC Bioinformatics*, 9:57, 2008.

97. X.M. Zhao, L. Chen, and K. Aihara. Protein classification with imbalanced data. *Proteins*, 70:1125–1132, 2008.

98. X.M. Zhao, Y. Cheung, and D.S. Huang. A novel approach to extracting features from motif content and protein composition for protein sequence classification. *Neural Networks*, 18:1019–1028, 2005.

99. X.M. Zhao, Y. Wang, L. Chen, and K. Aihara. Protein domain annotation with integration of heterogeneous information sources. *Proteins: Structure, Function, Bioinformatics*, 72:461–473, 2008.

100. X.M. Zhao and L. Chen. A new balanced ensemble classifier for predicting fungi protein subcellular localization based on protein primary structures. *The 2nd International Conference on BioMedical Engineering and Informatics* (BMEI'09), 2009.

101. C. Sun, W. Tang, L. Chen, and X.M. Zhao. Protein Subcellular Localization Prediction for Fusarium graminearum. *The 3rd International Symposium on Optimization and Systems Biology*, 2009, pp. 254–260.

102. X. Zhou, M.C.J. Kao, H. Huang, A. Wong, J. Nunez-Iglesias, M. Primig, O.M. Aparicio, C.E. Finch, T.E. Morgan, and W.H. Wong. Functional annotation and network reconstruction through cross-platform integration of microarray data. *Nature Biotechnol*, 23:238–243, 2005.

103. X. Zhou, M.C.J. Kao, and W.H. Wong. From the cover: Transitive functional annotation by shortest-path analysis of gene-expression data. *Proc Natl Acad Sci U S A*, 99:12783–12788, 2002.

# 23

# PROTEIN DOMAIN BOUNDARY PREDICTION

Paul D. Yoo, Bing Bing Zhou, and Albert Y. Zomaya

## 23.1 INTRODUCTION

The accurate delineation of protein domain boundaries is an important step for the prediction of protein structure, function, evolution, and design. Because a single domain spans an entire polypeptide chain or a subunit of such a chain, domains provide one of the most useful sources of information for understanding protein function, analysis based on domain families, and the study of individual proteins [1, 2].

Proteins comprise smaller building blocks, which are called “domains” or “modules.” These building blocks are distinct regions in a three-dimensional (3D) structure resulting in protein architectures assembled from modular segments that have evolved independently [3]. The modular nature of proteins has many advantages, offering new cooperative functions and enhanced stability. For example, new proteins, such as chimeric proteins, can be created because they consist of multifunctional domains [4]. The search method for templates used in comparative modeling can be optimized by delineating domain boundaries because the templates are classified based on domains [5]. Domain boundary prediction can improve the performance of threading methods by enhancing their signal-to-noise ratio [6] and, for homologous domains, plays a key role in reliable multiple sequence alignment [7].

During the past three decades, numerous methods using the 3D coordinates of a protein structure have been proposed for more accurately delineating domain boundaries [8]. However, in recent literature, there has been a major demand for fully automated approaches to identify domains in globular proteins from one-dimensional

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

(1D) atomic coordinates [9, 10]. This demand has significantly grown over recent years because genome and other sequencing projects have produced a flux of DNA and protein sequence data [11]. Most recent sequence-based methods have been built based on various machine learners because of their widely known learning ability.

DOMpro [12] uses evolutionary information (gene-exon shuffling), secondary structure, and solvent accessibility information with a recursive neural network; CHOPnet [13] uses evolutionary information, amino acid composition, and amino acid flexibility analyzed with a neural network; SnapDRAGON [14] predicts domain by using an *ab initio* protein folding method; DomSSEA [15] uses predicted secondary structure; Nagarajan and Yona's [16] method is based on analyzing multiple sequence alignments from a database search, position-specific physiochemical properties of amino acids, and predicted secondary structures analyzed with a neural network; SSEP-Domain [17] predicts domains by combining information of secondary structure element alignments, profile–profile alignments, and pattern searches; Armidillo [18] uses amino acid composition to predict domain boundaries; DomCut [19] uses a linker index deduced from a dataset of domain/linker segments; PRODO [20] uses a neural network method; and DomainDiscovery [21] uses support vector machines (SVMs) from sequence information including a domain linker index. These methods only can be successful in identifying domains if the sequence has detectable similarities to other sequence fragments in databases or when the length of unknown domains do not substantially deviate from the average of known protein structures. Many of these methods focus exclusively on predicting boundaries for two-domain chains. The overall accuracy of sequence-based methods has been reported in the range of 50–70% for single-domain proteins and considerably less (<40%) when limited to multidomain proteins [18, 22].

Many automated systems have shown reasonable improvements because they successfully have captured the information of a single molecule or of neighboring residues involving short-range (local) interactions. However, at the same time, their limitations in the exploitation of information from long-range (nonlocal) interactions have been observed [23–26]. These limitations are related to model overfitting and the weak signal-to-noise ratio associated with nonlocal interactions, which lead to the problem of the "vanishing gradient."

In this chapter, we introduce the novel interrange interaction integrated approach for protein domain boundary prediction with a brief overview of relevant literature. Our novel approach involves (i) the design of modular kernel algorithm, which can exploit effectively the information of nonlocal interactions, and (ii) the development of a novel profile that can provide suitable information to the algorithm. One of the key features of this profiling technique is the use of multiple structural alignments of remote homologues to create extended sequence profiles and combines the structural information with suitable chemical information that plays an important role in protein stability. This profile can capture the sequence characteristics of an entire structural superfamily and extend a range of profiles generated from sequence similarity alone. The chapter proceeds as follows: Section 23.2 summarizes the literature on protein data encoding and profiling and introduces our novel *Evolutionary and Hydrophobicity* profile (EH-profile). Section 23.3 introduces a new modular kernel approach for the effective exploitation of nonlocal information as well as the issues in the vanishing gradient problem. Section 23.4 looks at the overall architecture of the novel interrange interaction integrated approach. Section 23.5 concludes the chapter.

## 23.2 PROFILING TECHNIQUE

A protein sequence contains the 20 letters of the amino acid alphabet. For example, $A = \{\mathrm{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}\}$. Statistical machine-learning methods cannot process a direct representation of nucleic or amino acid sequences. Thus, it is inevitable to use a suitable sequence encoding method for the classifiers to recognize underlying regularities of amino acids [27–30].

Orthogonal encoding (also known as the binary representation and the distributed method) is one of the most commonly used amino acid encoding methods in the past decade. In orthogonal encoding, each letter is represented by a 20-dimensional binary vector indicating the presence of a particular amino acid type [31]. The 20 standard amino acids are ordered 1 through 20, and the $i$th amino acid has the binary codeword of 20 bits with the $i$th bit set to “1” and all others set to “0,” for $i = 1, 2, \dots, 20$ [32]. For example, *Alanine* is expressed by 10000000000000000000, and “Cysteine” is expressed by 01000000000000000000. One advantage is that it does not introduce any artificial correlations between the amino acids [33]. However, because the dimension of residue vector increases rapidly as $n$ increases, it may lead to a large computational cost and to model complexity (a typical input window of 13 residues requires $567 = (21 \times (2 \times 13 + 1)$ input layers and connecting weights), and recognition bias. Thus, it can cause poor performance of the classifiers [34–36].

The BLOSUM62 matrix is a more popularly used encoding method in recent years. It originally was designed by Henikoff and Henikoff [37] and also is known as an evolutionary information/profile. This is a measure of difference between two distantly related proteins. The blocks are found in multiple sequence alignment, and they are assumed to be of functional importance within related proteins. The BLOSUM62 indicates the matrix, calculated from observed substitutions between proteins that share 62% sequence identity or less. It has shown its superiority in detecting similarities in distant sequences, and this is the matrix used by default in most recent alignment applications such as BLAST and PSI-BLAST [38]. In the last decade, several machine-learning-based systems have used evolutionary profiles that contain homology information from sequence alignments and showed a striking improvement.

To construct the novel EH-profile, we use one effective hydrophobicity information (SARAH1 scale) in addition to the evolutionary information generated by PSI-BLAST. Several researchers selected hydrophobicity as the main feature among many other physicochemical properties for protein structure prediction (such as polarity, charge, or size) [39–41]. Several recent studies reported that the level of phosphorylation affect protein’s hydrophobicity significantly or vice versa [42, 43]. Hydrophobicity is a major factor in protein stability. The “hydrophobic effect” plays a fundamental role in the spontaneous folding of proteins. It can be expressed as

**Table 23.1 Hydrophobicity scale: nonpolar → polar distribution of amino acids chains, pH7 (kcal/mol)**

| | Amino Acid | Feature Value | | Amino Acid | Feature Value |
|---|---|---|---|---|---|
| 1 | I | 4.92 | 11 | Y | –0.14 |
| 2 | L | 4.92 | 12 | T | –2.57 |
| 3 | V | 4.04 | 13 | S | –3.40 |
| 4 | P | 4.04 | 14 | H | –4.66 |
| 5 | F | 2.98 | 15 | Q | –5.54 |
| 6 | M | 2.35 | 16 | K | –5.55 |
| 7 | W | 2.33 | 17 | N | –6.64 |
| 8 | A | 1.81 | 18 | E | –6.81 |
| 9 | C | 1.28 | 19 | D | –8.72 |
| 10 | G | 0.94 | 20 | R | –14.92 |

the free energy (kilocalories per mole) of transfer of amino acid side chains from cyclohexane to water. The amino acids with positive values of free energy in transferring cyclohexane to water are hydrophobic, and the ones with negative values are hydrophilic [39]. Table 23.1 shows hydrophobicity scales, and the hydrophobicity matrix can be formulated using the following function.

Given:

$$\text{Amino\_Acid[ ]} = \{\text{C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V}, \ldots\} \text{ and}$$
$$\text{Hydrophobicity\_Index[ ]} = \{1.28, -5.54, -6.81, 0.94, -4.66, 4.92, 4.92, -5.55, 2.35, 2.98, 4.04, -3.40, -2.57, 2.33, -0.14, 4.04, \ldots\},$$

$$\text{hydrophobicity\_matrix}[i][j] = \frac{\text{abs(hydrophobicity\_index}[i] - \text{hydrophobicity\_index}[j])}{20}$$

where the denominator 20 is used to convert the data range into [0,1]. The hydrophobicity matrix [3][4] means the absolute value of the difference of the hydrophobicity indices of two amino acids E (–6.81) and G (0.94). With the range adjustment, we obtain 0.2935.

For structure/function families and the classification of protein sequences, various hydrophobicity scales were examined thoroughly by David [44]. He reported the effectiveness of each hydrophobicity scale and concluded that the Rose scale [45] was superior to all others when used for protein structure prediction. The Rose scale is correlated to the average area of buried amino acids in globular proteins (Table 23.2). However, Korenberg *et al.* [46] pointed out several key drawbacks with the Rose scale. For example, because it is not a one-to-one mapping, different amino acid sequences can have identical hydrophobicity profiles; the scale covers a narrow range of values while causing some amino acids to be weighted more heavily than others. To overcome these problems, the SARAH1 scale—five bits "state" representation for amino acid—was introduced by Korenberg *et al.*

**Table 23.2 Rose hydrophobocity scale**

| | Amino Acid | Feature Value | | Amino Acid | Feature Value |
|---|---|---|---|---|---|
| 1 | A | 0.74 | 11 | L | 0.85 |
| 2 | R | 0.64 | 12 | K | 0.52 |
| 3 | N | 0.63 | 13 | M | 0.85 |
| 4 | D | 0.62 | 14 | F | 0.88 |
| 5 | C | 0.91 | 15 | P | 0.64 |
| 6 | Q | 0.62 | 16 | S | 0.66 |
| 7 | E | 0.62 | 17 | T | 0.70 |
| 8 | G | 0.72 | 18 | W | 0.85 |
| 9 | H | 0.78 | 19 | Y | 0.76 |
| 10 | I | 0.88 | 20 | V | 0.86 |

SARAH1 assigns each amino acid a unique five-bit signed code, where exactly two bits are nonzero. SARAH1 ranks 20 possible amino acids according to the Rose hydrophobicity scale. Each amino acid is assigned a five-bit code in descending order of the binary value of the corresponding code. One of the benefits of using the five-bit code is that the complexity of the classifier can be reduced significantly and can arrange these numbers in 32 possible ways ($2^5 = 32$). If the representations with no or all 1s, and those with one or four 1s are removed, then exactly 20 representations are left. This leaves just enough representation to code for the 20 amino acids. For a window size of 5, a residue vector has $5 \times 11 = 55$ dimensions, which leads to less model complexity than the residue vector using widely used orthogonal encoding ($20 \times 11 = 220$ dimensions) [47].

The resulting scale in Table 23.3, where the right-half is the negative mirror image of the left-half, is referred to as SARAH1. The 10 most hydrophobic residues are positive, and the 10 least hydrophobic residues are negative. Korenberg *et al.* indicated that although these scales carry information about hydrophobicity, scales similarly can be constructed to embed other chemical or physical properties of the amino acids such as polarity, charge, $\alpha$-*helical* preference, and residue volume.

**Table 23.3 SARAH1 Scale**

| | Amino Acid | Binary Code | | Amino Acid | Binary Code |
|---|---|---|---|---|---|
| 1 | C | 1,1,0,0,0 | 11 | G | 0,0,0,−1,−1 |
| 2 | F | 1,0,1,0,0 | 12 | T | 0,0, −1,0, −1 |
| 3 | I | 1,0,0,1,0 | 13 | S | 0,0, −1, −1,0 |
| 4 | V | 1,0,0,0,1 | 14 | R | 0, −1,0,0, −1 |
| 5 | L | 0,1,1,0,0 | 15 | P | 0, −1,0, −1,0 |
| 6 | W | 0,1,0,1,0 | 16 | N | 0, −1, −1,0,0 |
| 7 | M | 0,1,0,0,1 | 17 | D | −1,0,0,0, −1 |
| 8 | H | 0,0,1,1,0 | 18 | Q | −1,0,0, −1,0 |
| 9 | Y | 0,0,1,0,1 | 19 | E | −1,0, −1,0,0 |
| 10 | A | 0,0,0,1,1 | 20 | K | −1, −1,0,0,0 |

### 23.2.1 Nonlocal Interaction and Vanishing Gradient Problem

In a protein structure prediction problem, existing large kernel algorithms such as neural networks have performed well; however, they also have shown several limitations especially when dealing with nonlocal interactions in amino acids. The main difficulty with this class of neural networks is from the lack of generally efficient algorithms for solving numerical optimization. In particular, error minimization is known to fail in the presence of nonlocal interactions [48,49]. Interesting remedies to this vanishing gradient have been suggested in the literature [50,51]; however, their effectiveness in realistically large-scale supervised learning tasks has not been elucidated so far.

To overcome this limitation, one should be able to minimize the problem of the "vanishing gradient" [48,49]. For nonlocal interaction, residues that are close in space (3D strucure) occupy distant positions in the sequence. At each sequence position, the model may receive important structural information needed at distantly located sequences. Therefore, it must deal with long-term dependencies, which leads to the problem of the vanishing gradient. The vanishing gradient addresses the characteristics of nonchaotic dynamic systems that the gradient of states, with respect to previous states, vanishes exponentially with the temporal distance between these states. This feature of nonchaotic systems results from the fact that initial conditions do not have a large influence over later states. Therefore, nonchaotic systems are prevented from learning to store information over time.

This new modular approach to neural networks combines several methods and procedures to exploit effectively nonlocal information. The first step was to develop a modular kernel model and train it to predict domain boundaries of proteins with an EH-profile. Within this model, each kernel has a learning ability capable of bridging intervals of time even for noisy, incompressible input sequences, without the loss of a short-time-lag capability. Its architecture enforces constant error flow (thus, neither exploding nor vanishing) through internal states of units. Being modular, this approach requires several small networks to cooperate and communicate with each other to obtain the complete map of intermolecular interactions.

These networks comprise modules that can be categorized both according to their distinct structure and to their functionality, which are integrated together via an integrating unit. With functional categorization, each module is a neural network, which carries out a distinctly identifiable subtask. This approach allows different types of learning algorithms (these can be neural network based or otherwise) to be combined in a seamless fashion. Through the use and integration of the best-suited learning algorithms for a given task, there is a distinct improvement in artificial neural network learning. As with other modular learning systems, the main advantages include extendibility, incremental learning, continuous adaptation, economy of learning and relearning, and computational efficiency.

### 23.2.2 Hierarchical Mixture of Experts

This approach incorporates the hierarchical mixture of experts (HME), a well-known tree-structured model for regression and classification based on soft probabilistic splits of the input space [52]. In this model, the distribution of target variables is given by a mixture of component distributions in which the components, as well as the mixing coefficients, are conditioned on the input variables. The component distributions are referred to as *experts*, whereas mixing coefficients are controlled by *gating distributions*. Values for the parameters of this model can be set using an efficient EM algorithm to predict maximum likelihood [52]. The resulting model *automatically* will perform a soft partitioning of the dataset into groups corresponding to different regions of input space and simultaneously fit separate models (corresponding to the mixture components) to each of those groups.

The fundamental concept behind the probabilistic interpretation of this network is that a paralinguistic mapping of input vectors $x^{(t)}$ to output vectors $y^{(t)}$ in the dataset can be subdivided into a sequence of nested decisions, generating a probabilistic tree. For a particular input vector $x^{(t)}$, values generated by the gating networks are assumed to be multinomial probabilities that select one of the connected expert networks. A sequence of decisions starts from the top node influenced by the probability distributions of the intermediate gating networks. The process eventually ends at a specific terminal expert network.

HME describes a conditional probability distribution over a vector $t$ of target variables conditioned on a vector $x$ of inputs. Consider the case of functional mapping learning of the type $\vec{Y} = f\left(\vec{X}\right)$ based on training dataset $\mathrm{T} = \left(x^{(t)}, y^{(t)}\right)$, $t = 0, \ldots, n$ with $\vec{X} = \{x_1, x_2, \ldots, x_n\}$ and a corresponding desired response $\vec{Y} = \{y_1, y_2, \ldots, y_n\}$. All networks, both experts and gating, receive the same input vector at the $t$th time instant, $x^{(t)}$. However, although the gating networks use this input to compute confidence level values for the outputs of the connected expert networks, the expert networks themselves use the input to generate an estimate of the desired output value. The outputs of the gating networks are scalar values and are a partition of unity at each point in the input space (*i.e.*, a probability set). Thus, consider the two-layered binary branching HME; each of the expert neural networks $(i, j)$ produces outputs $y_{ij}$ from the input vector $x^{(t)}$ according to the following relationship:

$$y_{ij} = f\left(x^{(t)}, \vec{W}_{ij}\right)$$

where $f$ is a neural network mapping using input $x^{(t)}$ and its corresponding weight matrix $\vec{W}_{ij}$. The input vector $x^{(t)}$ is considered to have an additional constant value to allow for network bias. The gating networks are generally linear. Because they perform multidirectional classification among the expert networks, the nonlinear output is chosen to be a *softmax* (short for *soft maximum*). The outputs of the gating network $g_i$ at the top level are computed according to:

$$g_i \frac{e^{\varsigma_i}}{\sum_k e^{\varsigma_k}} \text{ with } \varsigma_i = \vec{V}_i^T x^{(t)}$$

where $\vec{V}_i$ is the weight matrix associated with gating network $g_i$. Because of the special form of the *softmax* being nonlinear, the $g_i$'s are positive and sum up to one for each input vector $x^{(t)}$. They can be interpreted as the local conditional probability in that an input vector $x^{(t)}$ lies in the affiliated partitioned subregion of the associated expert network. The lower level gating networks compute their output activations similar to the top level gating network according to the following expression:

$$g_{j|i} = \frac{e^{\varsigma_{ij}}}{\sum_k e^{\varsigma_{ik}}} \text{ with } \varsigma_{ij} = \vec{V}_{ij}^T x^{(t)}$$

The output activations of the expert networks are weighted by the gating networks' output activations as they proceed up the tree to form the overall output vector. Specifically, the output of the $i$th internal node in the second layer of the tree is:

$$y_i = \sum_j g_{j|i} y_{ij}$$

whereas the output at the top level node is:

$$y^{(t)} = \sum_i g_i y_i$$

Because both the expert and the gating networks compute their activations as functions of the input $\vec{X}$, the overall output of the architecture is a nonlinear function of the input.

### 23.2.3 Overall Modular Kernel Architecture

In this experiment, we use Benchmark_3 dataset. Benchmark_3 is a newly developed comprehensive dataset for benchmarking structure-based domain-identification methods. Benchmark_3 is similar to the dataset published by Holland *et al.* [53]; it contains proteins of known structures for which three methods (*e.g.,* CATH [54] and SCOP [55]) agree on the assignment of the number of domains. The dataset consists of 271 polypeptide chains, 106 one-domain chains (39.1%), 108 two-domain chains (39.9%), 45 three-domain chains (16.6%), seven four-domain chains (2.6%), and five-domain chains (1.9%). Also, 44 chains were removed from the Benchmark_2, as the overlap between the domains was below 90%. The dataset is nonredundant in a structural sense; each combination of topologies occurs only once per dataset. Sequences of protein chains are taken from the *Protein Data Bank* (PDB) [56]. The secondary-structure information and solvent accessibility are predicted for each chain in Benchmark_3, using SSpro [57] and ACCpro [58]. Evolutionary information for each chain is obtained using the *position-specific scoring matrix* (PSSM), which was constructed using PSI-BLAST.

Our modular approach contains three main components. First, given amino acid sequences, PSI-BLAST was used to generate PSSMs with an e-value threshold, for the inclusion of 0.001 and six search iterations of a nonredundant (nr) sequence database. The PSSM has 20 × $N$ elements, where $N$ is the length of the target sequence, and each element represents the log likelihood of a particular residue substitution based on a weighted average of BLOSUM62 matrix scores for a given alignment position in the template. Second, SARAH1 scales were computed from the amino acid chains in the Benchmark_3 dataset and combined with the PSSM. The EH-profile, which contains PSSMs and SARAH1 scales, all were normalized to fall in the interval [–1, 1] using the following algorithm:

$$pn = 2 \times (p - \min p)/(\max p - \min p) - 1$$

where $p$ is an $R \times Q$ matrix of input vectors, min$p$ is an $R \times 1$ vector containing minimums for each $p$, and max$p$ is $R \times 1$ vector containing maximums for each $p$.

Third, our modular kernel model used the resulting profile and performed its classification tasks. As discussed, we adopted a seven-fold cross-validation scheme for its evaluation. With the threshold $T$, the final predictions were simulated from the raw output generated by HME. During the postprocessing of the network output because the network generates raw outputs with many local peaks, we again adopted Liu and Rost's [13] method to filter the raw outputs. First, we determined the threshold for each network output according to the length ($L$) of the protein and to the distribution of raw output values for all residues in that protein. We compiled the 92nd percentile of the raw output $T_1$ and set the threshold $T$ to:

$$T = \begin{bmatrix} \max(T_1, 60) & \text{for } L \leq 100 \\ \max(T_1, 30) & \text{for } L \leq 200 \\ T_1 & \text{for } L > 200 \end{bmatrix}$$

$T$ was set to the threshold that divides domain boundaries and others. If the value of a residue was above the threshold, then the residue was regarded as a domain boundary. Second, we assigned the central residue as a domain boundary if three or more residues were predicted as a domain boundary. And all parameters for these filters were developed using the validation set alone.

The performance of each model was measured by the fractions of true negative and true positive ($TN_f$: the proportion of true negative data correctly predicted and $TP_f$: the proportion of true positive data correctly predicted), the sensitivity (Sn: the proportion of correctly predicted domain boundary residues with respect to the total positively identified residues), the specificity (Sp: the proportion of incorrectly predicted boundary residues with respect to the total number of domain boundary residues), correlation-coefficient (Cc: an equal balance between positive and negative predictions, between –1 and 1), and accuracy (Ac: the proportion of true-positive and true-negative residues with respect to the total positive and negative residues). Cc reflects a situation in which a method, that predicts every residue to be positive shows a prediction accuracy of 100% in detecting positive boundaries but

0% accuracy for negative residues. Hence, a high value of Cc means that the model is regarded as a more robust prediction system.

$$\mathrm{Sn} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}'} \quad \mathrm{Sp} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}'} \quad \mathrm{Ac} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}'}$$

and

$$\mathrm{Cc} = \frac{(\mathrm{TP} \times \mathrm{TN}) - (\mathrm{FN} \times \mathrm{FP})}{\sqrt{(\mathrm{TP} + \mathrm{FN}) \times (\mathrm{TN} + \mathrm{FP}) \times (\mathrm{TP} + \mathrm{FP}) \times (\mathrm{TN} + \mathrm{FN})}}$$

The stepwise procedure we have performed can be summarized as follows:

1. Data collection, building Benchmark_3 and preprocessing datasets
2. Profile construction, such as PSSM, Sarah1, and EH-profile
3. A hold-out method to divide the combined dataset into seven subsets (training and testing sets)
4. The information obtained in steps 2 and 3 were combined and normalized to fall in the interval [–1, 1] to be fed into networks
5. Target levels were assigned to each profile
   (a) Positive (1) for domain boundary residues and negative (–1) for nonboundary residues
6. Model training on each set to create a model
7. Simulation of each model on the test set to obtain predicted outputs
8. Postprocessing to find predicted domain boundary locations

The procedure from steps 6 through 8 was performed iteratively until we obtained the most suitable kernel and the optimal hyperparameters for HME for the Benchmark_3 dataset.

## 23.3 RESULTS

To see the suitability of our proposed approach in domain boundary prediction, we have chosen the most widely adopted machine-learning models and profiles for comparison. Our experiment has three consecutive steps. First, we compare the performance of our modular neural network HME with two other well-regarded machine-learning models in protein domain boundary prediction, transductive SVM and *multilayered perceptron* (MLP). Second, in the model comparison, the effectiveness of hydrophobicity information presented in the EH-profile is tested thoroughly and compared with widely known evolutionary profile, PSSM generated by PSI-BLAST [16]. Last, the performance of our *modular kernel approach* (MKA) that consists of an HME model and an EH-profile is compared with widely known protein domain boundary predictors.

**Table 23.4 Predictive performance of machine-learning models**

| Models | $TN_f$ | $TP_f$ | Sensitivity | Specificity | Correlation-Coefficient (Cc) | Accuracy |
|---|---|---|---|---|---|---|
| $HME_{HE}$ | 0.77 ± 0.015 | 0.79 ± 0.026 | 0.78 ± 0.002 | 0.78 ± 0.012 | 0.56 ± 0.016 | 0.78 ± 0.015 |
| $HME_{PSSM}$ | 0.74 ± 0.019 | 0.74 ± 0.018 | 0.75 ± 0.010 | 0.73 ± 0.045 | 0.48 ± 0.023 | 0.74 ± 0.016 |
| $SVM_{HE}$ | 0.71 ± 0.008 | 0.73 ± 0.010 | 0.70 ± 0.003 | 0.74 ± 0.017 | 0.44 ± 0.011 | 0.72 ± 0.020 |
| $SVM_{PSSM}$ | 0.71 ± 0.004 | 0.67 ± 0.008 | 0.65 ± 0.012 | 0.72 ± 0.006 | 0.37 ± 0.007 | 0.69 ± 0.003 |
| $MLP_{HE}$ | 0.69 ± 0.009 | 0.72 ± 0.012 | 0.61 ± 0.027 | 0.75 ± 0.019 | 0.40 ± 0.013 | 0.70 ± 0.025 |
| $MLP_{PSSM}$ | 0.67 ± 0.017 | 0.71 ± 0.032 | 0.61 ± 0.013 | 0.76 ± 0.027 | 0.37 ± 0.022 | 0.68 ± 0.011 |

Mean testing data ± standard deviation was obtained using the analysis of variance test using optimal settings for each model.

We adopted a seven-fold cross-validation scheme for the model evaluation in which random dataset selection and testing was conducted seven times. When multiple random training and testing experiments were performed, a model was formed from each training sample. The estimated prediction accuracy is the average of the prediction accuracy for the models, derived from the independently and randomly generated test divisions. In our preliminary experiments [59], we tested five different window sizes (3, 7, 11, 19, and 27) for each model and found that the window size of 11 is the most suitable for our experiments. (A window size of 11 means 23 amino acids with the boundary residue located at the center of the window.)

Table 23.4 summarizes confusion matrices for each test model. As indicated, the standard deviation for each model is insignificant, suggesting reasonable performance consistency. The average accuracy over three models for an EH-profile is about three percentage points better than an evolutionary profile. This proves our hypothesis that the hydrophobicity information used in an EH-profile provides suitable information as it performs key roles for protein stability. Clearly, an EH-profile is more useful than the widely known evolutionary profile for protein domain boundary prediction. More importantly, the performance of HME with an EH-profile showed the best predictive performance (Ac: 0.78). With an evolutionary profile, it also outperformed other models in Sn, Cc, and Ac. The modular approach used in HME improved its predictive performance by effectively capturing the information from nonlocal interactions. In other words, it is more resistant to model overfitting, and the weak signal-to-noise ratio associated, which lead to the problem of vanishing gradient.

Finally, our MKA that comprises the HME model and an EH-profile and two other well-known predictors were evaluated on the Benchmark_3 dataset. DOMpro [12] uses evolutionary information (gene-exon shuffling), secondary structure, and solvent accessibility information with a recursive neural network. DOMpro is trained and tested on a curated dataset derived from the CATH database. It achieved a sensitivity and specificity of 71% and 71%, respectively, in the CAFASP4 and was ranked among the top *ab initio* domain predictors. DomNet [59] is a recently introduced machine-learning algorithm that uses a novel compact domain profile (CD-profile). It outperformed nine other machine-learning methods on the Benchmark_2 dataset. DomNet is trained with an interdomain linker-region index, secondary structure, and

**Table 23.5 Accuracy of domain boundary placement on the Benchmark_3 dataset**

| Predictors | 1-Domain (39.1%) | 2-Domain (39.9%) | 3-Domain and More (21%) | Avg. Ac |
|---|---|---|---|---|
| **MKA** | 82.1% | 50.9% | 31.5% | 77.9% |
| **DomNet** | 83.0% | 54.3% | 21.0% | 78.6% |
| **DOMpro** | 79.2% | 48.1% | 29.8% | 74.2% |

relative solvent accessibility information with a CD-profile. The CD-profile uses additional structural information from a conserved-domain database [60] because conserved domains contain conserved sequence patterns or motifs, which allows for their detection in polypeptide sequences. Hence, the PSSMs in a conserved domain database can be useful to find remote homology.

Table 23.5 shows the accuracies obtained by each predictor on the Benchmark_3 dataset. MKA correctly predicted 86 of all 106 targets for one-domain chains and showed 82.1% accuracy. The accuracy of MKA was 0.9 percentage points less than DomNet in one-domain prediction. In two-domain prediction, DomNet still performed better as it predicted 113 of all 208 chains correctly. However, with three-domain and more chains, only MKA correctly predicted with more than 30% accuracy. Its accuracy in this category was 10.5 and 11.7 percentage points higher than DomNet and DOMpro, respectively. This means MKA more consistently captures information from an EH-profile and eventually leads to model stability and robustness. Although it is well acknowledged that the model stability is a more important factor than the learning bias in predictive performance [61], several important issues should be taken into account to improve the performance of the proposed MKA. This will be discussed in the following section.

## 23.4 DISCUSSION

Although many machine-learning-based domain predictors have been developed, they have shown limited capability for multidomain proteins. Our approaches used in MKA were shown to be effective for multidomain proteins. The two experiments confirmed our hypothesis that MKA efficiently captures nonlocal interaction information while preserving accurate data modeling in domain-boundary prediction. However, as its prediction accuracy reaches only about 40% for multidomain and 82% for one-domain proteins, there is still much room for improvement. Some areas of possible improvement are discussed in this section.

### 23.4.1 Nonlocal Interactions in Amino Acids

As historical summaries have shown [31], many researchers have built successful secondary structure predictors using machine learners such as feed-forward neural networks and support vector machines with local input windows of 9–15 amino acids [62–65]. Over the years, the performance has improved steadily by about

1% per year. This improvement was possible because of increased training data and several additional techniques including (i) output filers to cleanup predictions, (ii) input profiles associated with homologous sequence alignments, and (iii) predictor ensembles. The main weakness of these approaches resides in the researchers' use of a local window that cannot capture nonlocal information such as that presented in β-sheets. This is partially corroborated because the β-sheet class always shows the weakest performance results. Substantially increasing the input window's size, however, does not seem to improve the performance. As long as we fully cannot capture information about the interaction of remote sequence positions, efficient learning for the long-range dependencies does not seem possible. The learner is given only a set of inputs and a serial order relation for them and must solve a difficult credit assignment problem to identify the interacting positions.

Our modular kernel approach using HME architecture consists of comparatively simple experts (specialists neural) and gating networks, organized in a tree structure. The basic functional principle behind this structure is the well-known technique called "divide and conquer." This technique solves complex problems by dividing them into simpler problems for which solutions can be obtained easily. These partial solutions then are integrated to yield an overall solution to the whole problem. Its architecture enforces constant error flow (thus, neither exploding nor vanishing) through internal states of units.

Many gradient-based machine learners solve their classification problem (*i.e.,* function approximation) by explicitly hard splitting the input space into subregions, such that only one single 'expert' is contributing to the overall output of the model. The "hard splits" of the input space make algorithms to be variance increasing, especially for higher dimensional input spaces where data is distributed sparsely. In contrast, HME architecture uses a soft splitting approach to partition the input space instead of hard splitting, as is the case in statistical models, allowing the input data to be present simultaneously in multiple subregions. In this case, many experts may contribute to the overall output of the network, which has a variance decreasing effect.

### 23.4.2 Secondary Structure Information

In the literature, protein secondary-structure information has been used widely for domain-boundary prediction, as it was shown to be useful for increasing prediction accuracy. Most interdomain regions consist of loops, whereas β-strands tend to form sheets that constitute the core of protein domains. The α-helices and β-sheets in proteins are relatively rigid units, and therefore, domain boundaries rarely split these secondary structure elements. The mutations at the sequence level can obscure the similarity between homologs. However, their secondary-structure patterns remain more conserved because changes at the structural level are less tolerated. The secondary-structure-alignment methods used in this study aim to exploit these conserved features to locate domain regions within secondary-structure strings. We obtained the secondary-structure information by one of the widely known secondary-structure predictors called SSpro [57]. However, there is one significant limitation; the best predictor still cannot reach the upper boundary of prediction accuracy. The best secondary-structure predictors show only about 75–80% accuracy. Clearly, the incorrectly predicted secondary structures are highly likely to lead to the incorrect delineation of domain boundaries. Although the predicted secondary information seems to be useful for current approaches, it may not be ideal if one attempts to reach better than 80% accuracy.

### 23.4.3 Hydrophobicity and Profiles

For prediction or classification tasks, it is well-known that finding the right features or information plays key roles in improving model performance. Our profiling method based on the assumption that hydrophobicity, a major factor in protein stability with a suitable homology information, can provide better information for its computational leaner proved to be successful. However, many more issues need to be investigated, as indicated in various alignments studies [31–34]. One of the examples is human intervention in the adjustment of automatic alignment. As is believed widely, domain expert intervention at (i) fold identification and (ii) readjustments multiple alignment levels significantly can improve its accuracy. In the literature, therefore, research to develop more biologically realistic profiles actively has been reported. This should prevent the current limitations of automated methods by allowing domain experts to interact with the computation to control the quality of analysis at processing stages.

### 23.4.4 Domain Assignment Is More Accurate for Proteins with Fewer Domains

In general, the prediction accuracy of sequence-based methods has been far smaller (<50%) for multidomain proteins. For example, Liu and Rost's [13] experiments on CATH and SCOP assigned domains to random subsets of 1187 proteins of known high- resolution structure and less than 10% sequence homology; they showed correct prediction of the number of domains (single and multi) in 69% of the cases. However, the accuracy for multidomain cases alone was only 38%. For the two continuous-domain proteins, the average accuracy of dbp prediction in different validation runs was 46–51% considering a prediction to be correct if it were in ±20 residues interval of the CATH- and SCOP-assigned boundaries.

Joshi [22] discussed the main reasons for the problems in deciphering the multidomain protein structures and his possible solutions. With experimental data, although the structure within a domain is fixed, the relative positioning of two domains within the same chain can vary. For this reason, and because protein structural domains are independent folding units, it is unusual to find single crystal structures containing more than one domain. Similarly, protein modeling by database searching, sequence alignment, and/or phylogenic analysis is performed better on a single domain rather than on a multidomain polypeptide. Hence, in most cases, the number of domains in a protein first should be identified to determine the locations of such domains on the primary chain before embarking on a standard method of protein-structure/function determination. The identification of linker regions connecting two distinct domains is also useful in finding domain-boundary locations; accordingly, several domain-boundary predictors employing domain linker information, such as DomCut and DomainDiscovery, showed a reasonably better predictive performance in domain-boundary prediction.

## 23.5 CONCLUSIONS

Our approach adopted modular HME that leverages evolutionary and hydrophobicity information in the form of profiles and also used predicted secondary structure and relative solvent accessibility. This was demonstrated in the three consecutive experiments in this study. The novel EH-profile that combines homology information with hydrophobicity from the SARAH1 scale was successful in providing more structural and chemical information. In addition, the modular approach adopted in HME proved to be effective in capturing information from nonlocal interactions. Each gradient-memory-based model in HME showed a learning ability to bridge time intervals at some level in the nonlocal interaction environment (this is the case of noisy and incompressible input sequences), without much loss of a short time lag capability. With the newly built Benchmark_3 dataset, our approach showed its usefulness, especially for multidomain chains.

The experimental result on the Benchmark_3 dataset is encouraging; however, more work is need to extend it to a broader range of applications. Thus, we currently are developing a novel interactive (human-in-the-loop) profiling that can provide information from more distantly related homology. Considering difficult modeling cases, our modeling approach with human-in-the-loop profiling techniques seems to be an extremely promising stepping stone toward fully automated modeling. These approaches will be suitable for the exploitations of interresidue and homology information because it is based on more biologically realistic objective functions or scoring models. For example, automated alignments can be used as a starting point; however, the best sequence alignment of tertiary structures comes from careful human intervention. In addition, the algorithm will be modified further to have an intelligent and adaptive learning ability that senses changes in the environment and situation as human experts continually optimize the profiles. This ability, of course will aid significantly in the identification of potential problems (*i.e.,* critical points) during the computation and then enable the incorporation of different methods and other available information from various sources to execute more effective decisions.

## REFERENCES

1. R.R. Copley, T. Doerksa, I. Letunica, and P. Borka. Protein domain analysis in the era of complete genomes. *FEBS Lett*, 513:129–134, 2002.
2. M. Levitt and C. Chothia. Structural patterns in globular proteins. *Nature*, 261:552–558, 1976.

3. L. Kong and S. Ranganathan. Delineation of modular proteins: Domain boundary prediction from sequence information. *Briefings Bioinformatics*, 5(2):179–192, 2004.
4. M. Suyama and O. Ohara. DomCut: Prediction of inter-domain linker regions in amino acid sequences. *Bioinformatics*, 19(5):673–674, 2003.
5. B. Contreras-Moreira and P.A. Bates. Domain fishing: A first step in protein comparative modelling. *Bioinformatics*, 18:1141–1142, 2002.
6. S.J. Wheelan, A. Marchler-Bauer, and S.H. Bryant. Domain size distributions can predict domain boundaries. *Bioinformatics*, 16:613–618, 2000.
7. J. Gracy and P. Argos. DOMO: A new database of aligned protein domains. *Trends Biochem Sci*, 23:495–497, 1998.
8. S. Veretnik and I.N. Shindyalov. *Computational Methods for Domain Partitioning in Protein Structures in Computational Methods for Protein Structure and Modeling*. Springer-Verlag, New York, 2006.
9. I. Friedberg, L. Jaroszewski, Y. Ye, and A. Godzik. The interplay of fold recognition and experimental structure determination in structural genomics. *Curr Opin Struct Biol*, 14:307–312, 2004.
10. R.L. Marsden, T.A. Lewis, and C.A. Orengo. Towards a comprehensive structural coverage of completed genomics: a structural genomics viewpoint. *BMC Bioinformatics*, 8:86, 2007.
11. P.D. Yoo, B.B. Zhou, and A.Y. Zomaya. Machine learning techniques for protein secondary structure prediction: an overview and evaluation, *J Curr Bioinformatics*, 3(2):74–86, 2008.
12. J. Cheng, M. Sweredoski, and P. Baldi. DOMpro: Protein domain prediction using profiles, secondary structure, relative solvent accessibility, and recursive neural networks. *Data Min Knowl Discov*, 13(1):1–10, 2006.
13. J. Liu and B. Rost. Sequence-based prediction of protein domains. *Nucleic Acids Res*, 32(12):3522–3530, 2004.
14. R.A. George and J. Heringa. SnapDRAGON: A method to delineate protein structural domains from sequence data. *J Mol Biol*, 316:839–851, 2002.
15. R.N. Marsden, L.J. McGuffin, and D.T. Jones. Rapid protein domain assignment from amino acid sequence using predicted secondary structure. *Protein Sci*, 11:2814–2824, 2002.
16. N. Nagarajan and G. Yona. Automatic prediction of protein domains from sequence information using a hybrid learning system. *Bioinformatics*, 20:1335–1360, 2004.
17. J.E. Gewehr and R. Zimmer. SSEP-Domain: Protein domain prediction by alignment of secondary structure elements and profiles. *Bioinformatics*, 22(2):181–187, 2006.
18. M. Dumontier, R. Feldman, H.J. Yao, and C.W.V Hogue. Armidillo: Doamin boundary prediction by amino acid composition. *J Mol Biol*, 350:1061–1073, 2005.
19. M. Suyama and O. Ohara. DomCut: Prediction of inter-domain linker regions in amino acid sequences. *Bioinformatics*, 19(5):673–674, 2003.
20. J. Sim, S.Y. Kim, and J. Lee. PRODO: Prediction of protein domain boundaries using neural networks. *Proteins*, 59:627–632, 2005.
21. A.R. Sikder and A.Y. Zomaya. Improving the performance of DomainDiscovery of protein domain boundary assignment using inter-domain linker index. *BMC Bioinformatics*, 7(Suppl 5):S6, 2006.

22. R.R. Joshi. A decade of computing to traverse the labyrinth of protein domains. *Curr Bioinformatics*, 2:113–131, 2007.
23. P. Baldi and G. Pollastri. A machine learning strategy for protein analysis. *Intell Syst Biol*, 17(2):28–35, 2002.
24. P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(1):937–946, 1999.
25. M. Punta and B. Rost. PROFcon: Novel prediction of long-range contacts. *Bioinformatics*, 21(13):2960–2968, 2005.
26. J. Chen and N.S. Chaudhari. Cascaded bidirectional recurrent neural networks for protein secondary structure prediction. *IEEE Trans Comput Biol Bioinformatics*, 4(4):572–582, 2007.
27. J.R. Green, M.J. Korenberg, R. David, and I.W. Hunter. Recognition of adenosine triphosphate binding sites using parallel cascade system identification. *Ann Biomed Eng*, 31:462–470, 2003.
28. J.T. Wang, Q. Ma, D. Shasha, and C.H. Wu. Application of neural networks to biological data mining: A case study in protein sequence classification. *Proceedings of the 6th ACM SIGKDD International Conference*, KDD, 2000, pp. 305–309.
29. G. White and W. Seffens. Using a neural network to backtranslate amino acid sequences. *EJB*, 7:157–182, 1998.
30. N. Zavaljevski, F.J. Stevens, and J. Reifman. Support vector machines with selective kernel scaling for protein classification and identification of key amino acid positions. *Bioinformatics*, 18(5):689–696, 2002.
31. P. Baldi and S. Brunak. *Bioinformatics-the Machine Learning Approach*. MIT Press, Cambridge, MA, 1998.
32. B. Zhang, C. Zhihang, and Y.L. Murphey. Protein secondary structure prediction using machine learning. *Proc IEEE Conf on Neural Networks*, 1:532–537, 2005.
33. S. Akkaladevi, A.K. Katangur, S. Belkasim, and Y. Pan. Protein secondary structure prediction using neural network and simulated annealing algorithm. *Engineering in Medicine and Biology Society 26th Annual International Conference*, 2:2987–2990, 2004.
34. K. Lin, A.C.W. May, and W.R. Taylor. Amino acid encoding schemes from protein structure alignments: Multi-dimensional vectors to describe residue types. *J Theor Biol*, 216:361–365, 2002.
35. R. Lohmann, G. Schneider, D. Behrens, and P. Wrede. A neural network model for the prediction of membrane-spanning amino acid sequences. *Protein Sci*, 3:1597–1601, 1994.
36. Z.R. Yang and K.C. Chou. Bio-support vector machines for computational proteomics. *Bioinformatics*, 20:735–741, 2004.
37. S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Nat Acad Sci*, 89:10915–10919, 1992.
38. S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, 1997.
39. H. Hu and Y. Pan. Improved protein secondary structure prediction using support vector machine with a new encoding scheme and an advanced tertiary classifier. *IEEE Trans nanobioscience*, 3(4):265–271, 2004.

40. H. Kim and H. Park. Protein secondary structure prediction based on an improved support vector machines approach. *Protein Eng*, 16:553–560, 2003.
41. M.J. Korenberg, R. David, I.W. Hunter, and J.E. Solomon. Automatic classification of protein sequences into structure/function groups via parallel cascade identification: A feasibility study. *Ann Biomed Eng*, 28(7):803–811, 2000.
42. H.H. Jang, S.Y. Kim, S.K. Park, H.S. Jeon, Y.M. Lee, J.H. Jung, S.Y. Lee, H.B. Chae, Y.J. Jung, K.O. Lee, *et al.* Phosphorylation and concomitant structural changes in human 2-Cys peroxiredoxin isotype I differentially regulate its peroxidase and molecular chaperone functions. *FEBS Lett* 580(1):351–355, 2006.
43. C. Li, H.R. Ibrahim, Y. Sugimoto, H. Hatta, and T. Aoki. Improvement of functional properties of egg white protein through phosphorylation by dry-heating in the presence of pyrophosphate. *J Agric Food Chem*, 52(18):5752–5758, 2004.
44. R. David. *Applications of Nonlinear System Identification to Protein Structural Prediction*. Master's Thesis, MIT, Cambridge, MA 2000.
45. G.D. Rose, A.R. Geselowitz, G.J. Lesser, R.H. Lee, and M.H. Zehfus. Hydrophobicity of amino acid residues in globular proteins. *Science*, 229:834–838, 1985.
46. M.J. Korenberg, R. David, I.W. Hunter, and J.E. Solomon. Automatic classification of protein sequences into structure/function groups via parallel cascade identification: A feasibility study. *Ann Biomed Eng*, 28(7):803–811, 2000.
47. B. Zhang, C. Zhihang, and Y.L. Murphey. Protein secondary structure prediction using machine learning. *Proc. IEEE Conf on Neural Networks*, 1:532–537, 2005.
48. S. Hochreiter, Y. Bengio, P. Frasconi, and S. Schmidhuber. Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In S.C. Kremer and J.F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, Piscataway, NJ, 2001.
49. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans Neural Networks*, 5(2):157–166, 1994.
50. S. Hochreiter, and J. Schmidhuber. Long shot-term memory. *Neural Comput*, 9:1735–1780, 1997.
51. F. Gers, N. Schraudolph, and J. Schmidhuber. Learning precise timing with LSTM recurrent networks. *Mach Learn Res*, 3:115–143, 2002.
52. M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Comp*, 6:181–214, 1994.
53. T.A. Holland, S. Veretnik, I.N. Shindyalov, and P.E. Bourne. Partitioning protein structures into domains: Why is it so difficult? *J Mol Biol*, 361(3):562–590, 2006.
54. F.M.G. Pearl, D. Lee, J.E. Bray, I. Sillitoe, A.E. Todd, A.P. Harrison, J.M. Thornton, and C.A. Orengo. Assigning genomic sequences to CATH. *Nucleic Acids Res*, 28(1):277–282, 2000.
55. A. Andreeva, D. Howorth, S.E. Brenner, T.J. Hubbard, C. Chothia, and A.G. Murzin. SCOP database in 2004: Refinements integrate structure and sequence family data. *Nucleic Acids Res*, 32:D226–D229, 2004.
56. H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindylov, and P.E. Bourne. The Protein Data Bank. *Nucleic Acids Res*, 28:235–242, 2000.
57. G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins*, 47:228–235, 2002.

58. G. Pollastri, P. Baldi, P. Fariselli, and R. Casadio. Prediction of coordination number and relative solvent accessibility in proteins. *Proteins*, 47:142–153, 2002.

59. P.D. Yoo, A. Sikder, J. Taheri, B.B. Zhou, and A. Zomaya. DomNet: Protein domain boundary prediction using enhanced general regression network and new profiles. *IEEE Trans Nanobioscience*, 7(2):172–181, 2008.

60. A. Marchler-Bauer, J.B. Anderson, P.F. Cherukuri, C. DeWeese-Scott, L.Y. Geer, M. Gwadz, S. He, D.I. Hurwitz, J.D. Jackson, Z. Ke, *et al.* CDD: A conserved domain database for protein classification. *Nucleic Acids Res*, 33:D192–D196, 2006.

61. T.G. Dietterich and G. Bakiri. Machine learning bias, statistical bias and statistical variance of decision tree algorithms. Technical Report, Department of Computer Science, Oregon State OR, University, Corvallies, 1995.

62. B. Rost and C. Sander. Combining evolutionary information and neural networks to predict protein secondary structure. *Proteins*, 19(1):55–72, 1994.

63. D.T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, 292(2):195–202, 1999.

64. K. Ginalski, A. Elofsson, D. Fischer, and L. Rychlewski. 3D-jury: A simple approach to improve protein structure predictions. *Bioinformatics*, 19(8):1015–1018, 2003.

65. J. Sim, S.-Y. Kim, and J. Lee. PRODO: Prediction of protein domain boundaries using neural networks. *Proteins*, 59:627–632, 2005.

24

# AN INTRODUCTION TO RNA STRUCTURE AND PSEUDOKNOT PREDICTION

Jana Sperschneider and Amitava Datta

*If 10% of protein fold researchers switched to RNA, the problem could be solved in one or two years.*

—Tinoco and Bustamante, 1999 [39]

## 24.1 INTRODUCTION

For nearly a century, the view most researchers had on RNA did not reach beyond the central dogma of molecular biology: DNA is transcribed to messenger RNA, which in turn is translated to proteins. Consequently, RNA was seen solely as the passive carrier of genetic information from DNA to proteins and not much else. In the 1980s, the field of RNA experienced major turbulences with the discovery of the ability of RNA to act as a catalyst. Numerous noncoding RNAs that are not translated to proteins have been discovered, and the prevailing view among scientists is that these functional RNAs no longer have to hide behind proteins. Unlike DNA with its famous double helix structure, RNA exhibits diverse three-dimensional folds and is an extremely versatile molecule. For example, RNA is known to take part in translational regulation, intron splicing, and gene expression. Novel functional RNAs are discovered continuously, and many more breakthroughs in this exciting area can be expected in the foreseeable future.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

One of the central paradigms in molecular biology states that structure is related strongly to function. Laboratory structure prediction is intricate and costly. Therefore, computational structure prediction from sequence is one of the holy grails in bioinformatics. The most critical point in structure prediction is defining a reasonable computational model for the underlying complex biological folding processes. It is no trivial task to achieve a good balance between computational feasibility and realistic biological modeling. RNA folding is a complex process that is driven by base pair formation, loop entropy, temperature, and contact with other macromolecules and metal ions. To make things even more complicated, certain types of RNA have been found to adopt alternative conformations. However, the game is not necessarily lost. The main driving force in RNA folding allows for computational prediction; a continuous pairing of complementary bases stabilizes RNA structures through hydrogen bonds, resulting in a so-called *secondary structure*.

In the following sections, we will introduce algorithms for RNA structure prediction given a single sequence. Prediction of the consensus structure for several related RNA sequences is another challenge that will not be covered here. Comparative information can improve greatly structure prediction; however, existing algorithms are expensive computationally and need a reliable set of homologous sequences [15].

## 24.2 RNA SECONDARY STRUCTURE PREDICTION

RNA primary structure is a single-stranded chain of nucleotide monomers. Each nucleotide consists of a five-carbon sugar, a phosphate, and one of the four different bases: adenine (A), cytosine (C), guanine (G), or uracil (U). The phosphate is attached to the third and fifth carbon in the sugar, forming a backbone with certain directionality. By convention, an RNA sequence is written as the succession of its bases in $5'$ to $3'$ direction. Complementary bases can form so-called *canonical base pairs* (C,G), (A,U), or (G,U) through hydrogen bonding (Figure 24.1).

The structure that forms through base pairing of the primary sequence is called the *secondary structure*. Formally, an RNA secondary structure is defined as a set of base pairs in which each base is paired at most once. Given an RNA sequence $S = S_1 \dots S_n$ with $S_i \in \{\mathrm{A, C, G, U}\}$, its secondary structure $R$ is the set of base pairs

$$R = \big\{(i, j) \mid 1 \leq i < j \leq n \wedge S_i \text{ and } S_j \text{ form a base pair}\big\} \tag{24.1}$$

with the property that for all base pairs $(i, j), (i', j') \in R$ the following holds:

$$i = i' \Leftrightarrow j = j' \tag{24.2}$$

However, base pairing is not the only force in secondary structure formation. Many bases remain unpaired and form unstructured regions, which are referred to as *loops*.

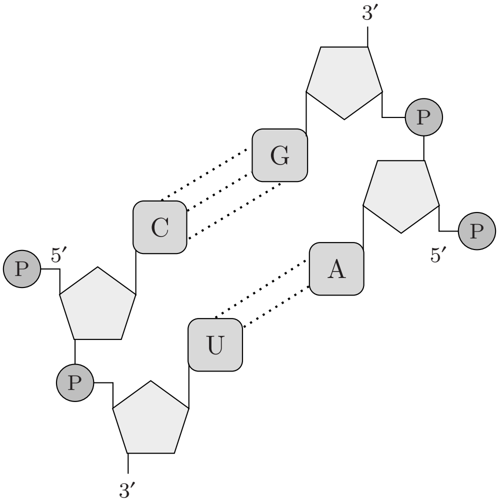


**Figure 24.1** RNA consists of a sugar-phosphate backbone with bases A, C, G, or U attached to the first carbon in the sugar. Complementary bases form pairs through stable hydrogen bonds. The pair (C,G) forms three hydrogen bonds, whereas (A,U) exhibits two.

The secondary structure for an RNA sequence can be decomposed into *structure elements*, as shown in Figure 24.2. Secondary structure elements typically are defined through their closing base pair (*i.e.*, the bond with largest distance). The following describes variations of secondary structure elements:

- Two or more consecutive base pairs are called a *helix* or *stem*.
- A loop with one closing base pair is called a *hairpin loop*.
- A loop with two closing base pairs that interrupt one strand of a helix is called a *bulge loop*.

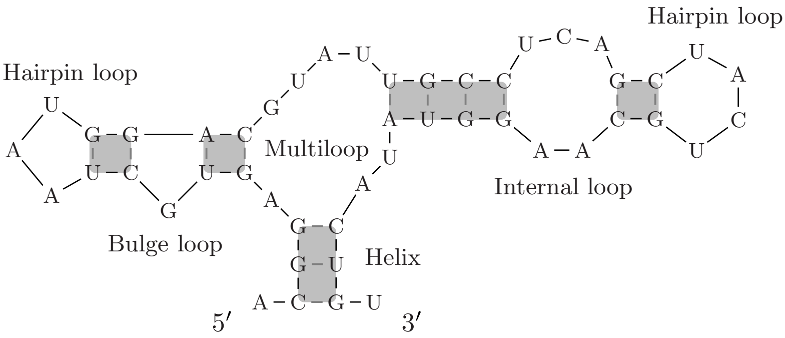


**Figure 24.2** Secondary structure elements in an RNA structure.

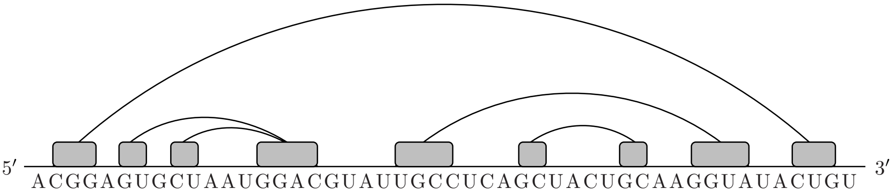


**Figure 24.3** Secondary structure elements as noncrossing intervals on the line.

- A loop with two closing base pairs that interrupt both strands of a helix is called an *internal loop*.
- A loop with one closing base pair and two or more interior base pairs is called a *multiloop*.

One major restriction commonly is imposed on RNA secondary structures. The set of base pairs $R$ has to form a *noncrossing* or *nested structure* such that for every two base pairs $(i, j), (k, l) \in R$ with $i < k$ the following holds:

$$i < j < k < l \text{ or } i < k < l < j \tag{24.3}$$

The structure displayed in Figure 24.2 also can be drawn in a more abstract way as a set of paired intervals on the line. This representation emphasizes that all occurring secondary structure elements are noncrossing (Figure 24.3).

However, crossing base pairs are important as they may result in the formation of a functional structure element, a so-called *pseudoknot*. Therefore, a noncrossing structure $R$ also is called *free of pseudoknots*. Despite their abundance, pseudoknots are for the most part neglected in computational RNA structure prediction as they significantly increase computational runtime. We will introduce practical structure prediction algorithms including pseudoknots in Section 24.3.

### 24.2.1 Minimum Free Energy Model

Complementary base pairing is the key to RNA secondary structure formation with the goal to minimize the free energy of the system. Out of the exponential number of possible structures, an RNA molecule will fold into the one with minimum free energy. The free energy change $\Delta G$ always is determined by both enthalpic and entropic forces. In RNA structures, enthalpic terms develop from consecutive base pairs (stabilizing forces), and entropic terms develop from unstructured regions (destabilizing forces).

Today, the most popular and robust model for RNA structure prediction is the minimum free energy (MFE) model. It includes free energy parameters for helices and loops that have been derived experimentally from synthetically constructed oligoribonucleotides [42, 26]. This empirical energy model commonly is referred to as the *nearest-neighbor model*. In a helix, the free energy is determined by the sum of the

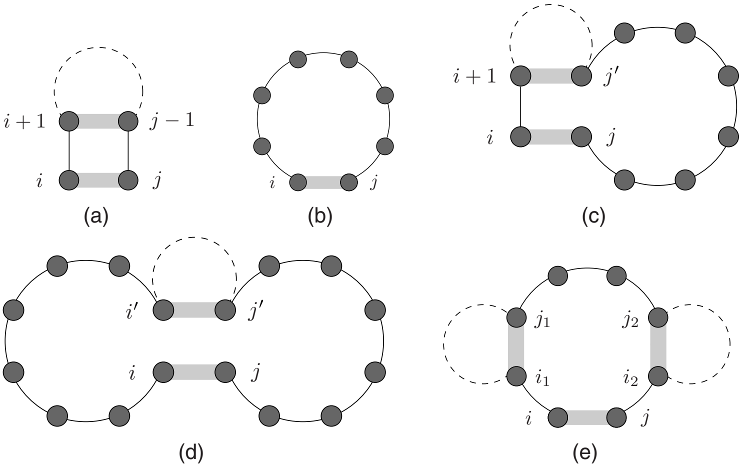


**Figure 24.4** RNA structure elements defined by the closing base pair $(i, j)$: (a) helix; (b) hairpin loop; (c) bulge loop; (d) internal loop; and (e) multiloop.

nearest-neighbor parameters for the stacked base pairs. The loop entropy parameters depend on the type of loop, the closing base pairs, and length. In the nearest-neighbor model, each secondary structure element is defined through its closing base pair $(i, j)$ (Figure 24.4). Each structure element has a specific energy function assigned to it as follows:

- A *helix* or *stem* with stacked base pairs $(i, j)$ and $(i + 1, j - 1)$ has free energy $eS(i, j)$.
- A *hairpin loop* with closing base pair $(i, j)$ has free energy $eH(i, j)$.
- A *bulge* or *internal loop* with closing base pairs $(i, j)$ and $(i', j')$ has free energy $eL(i, j, i', j')$.
- A *multiloop* with one closing base pair $(i, j)$ and $k$ interior base pairs $(i_1, j_1), \ldots, (i_k, j_k)$ has free energy $eM(i, j, i_1, j_1, \ldots, i_k, j_k)$.

A given RNA structure $R$ can be decomposed into the set of structure elements defined through their closing bonds. Let $E^R_{i,j}$ be the free energy of a structure element in $R$ closed by base pair $(i, j)$. The overall free energy $E(R)$ for an RNA structure $R$ is estimated by summing the energy parameters for each structure element in $R$:

$$E(R) = \sum_{(i,j)\in R} E^R_{i,j} \tag{24.4}$$

This can be done for two reasons. First, each structure element is defined uniquely through its closing bond $(i, j)$. Second, the free energy of a loop is assumed to be independent from all other loops in the nearest-neighbor model.

The nearest-neighbor energy model is used widely in algorithms for RNA secondary structure prediction from sequence. The key concept that allows for an efficient dynamic programming algorithm is that the energy of a structure element is self-contained, and the underlying energy model is additive. However, it should be emphasized that it is only an approximation of the underlying complex RNA folding process *in vivo*. An algorithm always is limited by the underlying model. Therefore, improving the free energy parameters surely will lead to better prediction results in the future.

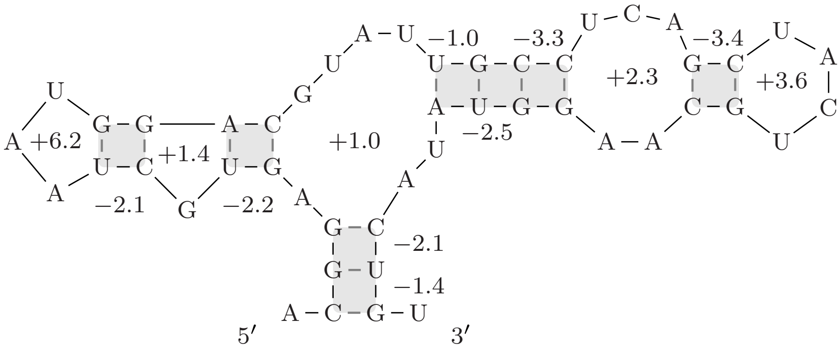
## ■ EXAMPLE 24.1

A toy example for the minimum free energy calculation under the nearest-neighbor model is shown in Figure 24.5. The values are taken from the set of free energy parameters often referred to as the Turner parameters [42, 26].

### 24.2.2 Prediction of Minimum Free Energy Structure

Computational RNA structure prediction started to develop in 1980 with a dynamic programming algorithm for calculating the structure with a maximum number of base pairs [28]. This obvious oversimplification of the folding process soon was overcome with the integration of free energy minimization and the nearest-neighbor energy model into a dynamic programming approach [44]. This pioneering work is still the most popular method for RNA structure prediction because of computational efficiency and mathematical correctness in regard to the underlying energy model.

For a given RNA sequence, how can we calculate a structure with minimum free energy? An exponential number of possible structures in the search space makes



$$
\begin{aligned}
\Delta G &= -1.4 - 2.1 - 2.2 + 1.4 - 2.1 + 6.2 + 1.0 - 1.0 - 2.5 - 3.3 + 2.3 - 3.4 + 3.6 \\
&= -3.5 \tfrac{\text{kcal}}{\text{mol}} \text{ at } 37^{\circ}\text{C}
\end{aligned}
$$

**Figure 24.5** The overall free energy for an RNA structure is the additive sum of free energy values for the secondary structure elements.

finding a minimum free energy structure by naive enumeration impossible. However, the Bellman principle allows for an efficient calculation: calculate the optimal solution for a structure from already computed optimal solutions for smaller substructures. The underlying energy model is based on the additivity of free energy values for secondary structure elements and consequently allows for dynamic programming. This enables us to compute a secondary structure with minimum free energy for a given RNA sequence in $O(n^3)$ time and $O(n^2)$ space. The algorithmic procedure based on [44] is described as follows:

**Input:** RNA sequence $S = S_1 \dots S_n$ with $S_i \in \{A, C, G, U\}$.
**Output:** Secondary structure $R$ with minimum free energy given as a set of base pairs $R = \{(i, j) \mid 1 \leq i < j \leq n \wedge S_i \text{ and } S_j \text{ form a base pair}\}$.

At the heart of the algorithm lies the computation of two possible scenarios for each subsequence $S = S_i \dots S_j$ using the matrices $W(i, j)$ and $V(i, j)$.

- $W(i, j)$ holds the free energy for the optimal secondary structure on the sequence $S = S_i \dots S_j$.
- $V(i, j)$ holds the free energy for the optimal secondary structure on the sequence $S = S_i \dots S_j$ *closed by base pair* $(i, j)$. If $(i, j)$ is not a valid base pair, then $V(i, j) = \infty$.

***24.2.2.1 Matrix V(i, j).*** The recursion scheme for matrix $V(i, j)$ is related directly to the set of possible secondary structure elements. The structure element with minimum free energy closed by base pair $(i, j)$ is either a helix, hairpin loop, bulge loop, internal loop, or multiloop. The recursion scheme for $V(i, j)$ is as follows:

$$V(i, j) = \min \begin{cases} eS(i, j) + V(i + 1, j - 1) \\ eH(i, j) \\ VL(i, j) \\ VM(i, j) \end{cases} \tag{24.5}$$

Here, two matrices $VL(i, j)$ and $VM(i, j)$ are introduced for the computation of bulge loops, internal loops, and multiloops with the requirement that $i$ and $j$ are base paired. For bulge and internal loops, one has to search for the best interior base pair $(i', j')$. Therefore, the following recursion holds:

$$VL(i, j) = \min_{\substack{i<i'<j'<j \\ i'-i+j-j'>2}} \left\{ eL(i, j, i', j') + V(i', j') \right\} \tag{24.6}$$

For a multiloop with a closing base pair $(i, j)$, one has to find the optimal $k$ interior base pairs $(i_1, j_1), \dots, (i_k, j_k)$ that minimize the energy function $eM(i, j, i_1, j_1, \dots, i_k, j_k)$. However, this obviously requires exponential runtime. Therefore, a simplified affine energy function is employed. The free energy of a multiloop is approximated as follows using the number of unpaired nucleotides in the loop, the number of interior base pairs in the loop, and constants $a$, $b$, and $c$. The parameters $a$, $b$, and $c$ describe the offset penalty, interior base pair penalty, and free base penalty, respectively.

$$\begin{aligned} &eM(i, j, i_1, j_1, \ldots, i_k, j_k) = \\ &a + bk + c\left(i_1 - i - 1 + j - j_k - 1 + \textstyle\sum_{l=1}^{k-1} (i_{l+1} - j_l - 1)\right) \end{aligned} \tag{24.7}$$

This modified energy function is the basis for an efficient calculation of matrix $VM(i, j)$. The calculation for matrix $VM(i, j)$ demands that the multiloop consists of at least two parts and the offset penalty $a$ is added.

$$VM(i, j) = \min_{i+1<k\leq j-1} \{WM(i + 1, k - 1) + WM(k, j - 1) + a\} \tag{24.8}$$

We need to ensure that the two parts $WM(i + 1, k - 1)$ and $WM(k, j - 1)$ contain at least one helix each. Remember that a multiloop is defined to have one closing base pair and two or more interior base pairs. This means that matrix $WM(i, j)$ needs to produce at least one interior base pair through its recursion scheme. Therefore, the initialization is as follows: $WM(i, i) = \infty$. In the calculation for $WM(i, j)$, the parts of the multiloop recursively are cut down to the interior base pairs, and the penalties are added accordingly.

$$WM(i, j) = \min \begin{cases} WM(i + 1, j) + c \\ WM(i, j - 1) + c \\ \min_{i<k\leq j} \{WM(i, k - 1) + WM(k, j)\} \\ V(i, j) + b \end{cases} \tag{24.9}$$

***24.2.2.2 Matrix W(i, j).*** Let us look at the subsequence $S = S_i \ldots S_j$ again. So far only structure elements with a closing base pair $(i, j)$ have been covered. However, three more cases can occur:

- Base $i$ remains unpaired.
- Base $j$ remains unpaired.
- Bases $i$ and $j$ are paired but not with each other.

These cases are captured in the following recursion equations.

$$W(i, j) = \min \begin{cases} W(i + 1, j) \\ W(i, j - 1) \\ V(i, j) \\ \min_{i<i'<j-1} \left\{W(i, i') + W(i' + 1, j)\right\} \end{cases} \tag{24.10}$$

We initialize $W(0, 0) = 0$. The final result can be read from $W(1, n)$, and the minimum free energy structure is recovered through a traceback step.

**24.2.2.3 Recursion Scheme.** The overall recursion scheme is displayed in Figure 24.6 and follows the notion of recursive diagrams used in [32, 11]. Here, a solid curved line between two bases indicates that they must form a base pair. A dashed curved line between two bases does not enforce them to pair. Shaded areas for matrix $V(i, j)$ denote a part of the sequence for which the secondary structure has been determined. Note that matrix $VL(i, j)$ is not shown explicitly; instead it has been incorporated into the recursions for $V(i, j)$.

**24.2.2.4 Time and Space Analysis.** For finding the minimum folding energy, one needs to compute matrix entry $W(1, n)$. The corresponding structure is recovered through a traceback step. This requires filling of all four matrices $W(i, j)$, $V(i, j)$, $VL(i, j)$, and $WM(i, j)$, which have $n^2$ entries each. As discussed before, the calculation of multiloops in $WM(i, j)$ is reduced to $O(n^3)$ time because of the affine energy function. Another critical part is the calculation of matrix $VL(i, j)$. Here, one needs to minimize over two biologically plausible positions $i'$ and $j'$, which leads to a runtime of $O(n^4)$. However, certain internal loop energy assumptions allow for a reduction of runtime of $O(n^3)$ [25]. Using these techniques, the overall time requirement is $O(n^3)$ using $O(n^2)$ space.

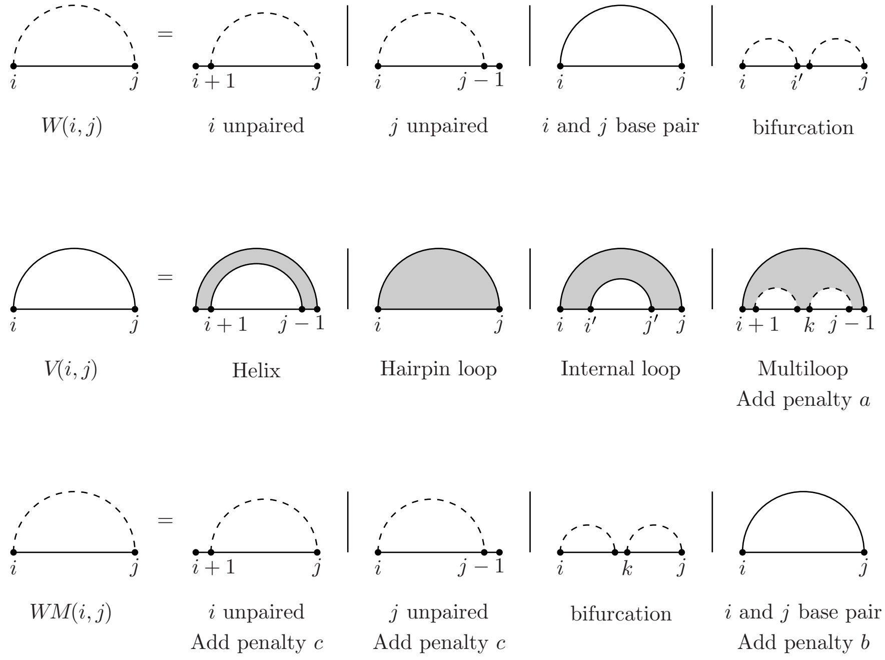


**Figure 24.6** Recursion scheme for the matrices used in the dynamic programming algorithm.

***24.2.2.5 Summary.*** Dynamic programming is today the most popular method for RNA structure prediction given a single sequence. However, minimum free energy folding under the nearest-neighbor model has its strengths and weaknesses, which are discussed in the following:

- The dynamic programming algorithm returns the correct mathematical solution for the underlying energy model. It is a very fast method and is therefore feasible for long sequences. However, the predicted minimum free energy structure might not be the native one. The nearest-neighbor energy parameters are only approximate, and inaccuracies lead to unreliable predictions, especially for longer sequences.
- Pseudoknots are important parts of RNA structures; however, secondary structure prediction methods often neglect them. The dynamic programming method for minimum free energy folding cannot accommodate pseudoknots under the same time and space requirements.
- Some RNAs are capable of forming alternative structures; therefore, prediction of a single minimum free energy structure is often inadequate.

The average accuracy of minimum free energy folding for a diverse set of sequences shorter than 700 nucleotides was reported to be about 70% in [26]. Researchers should be aware of the downsides of minimum free energy folding and ideally seek experimental verification. However, this is not always feasible, and thus, we will look at other computational techniques. The following section will deal with calculation of the partition function for improved prediction results through examination of the folding ensemble. Section 24.3 will examine how to include pseudoknots in RNA structure prediction.

### 24.2.3 Partition Function Calculation

The prediction of a structure with minimum free energy not only suffers from inaccuracies of the underlying energy model, but it also is designed to return only one optimal solution. Given a sequence, there is an exponential number of possible structures in the search space. Surprisingly, near-optimal structures within say 10% of the computed minimum free energy are abundant. The suboptimal structure space should not be neglected for two reasons. First, the correct result might be found among the suboptimal solutions as a result of approximate energy parameters. Second, a close examination of the structure space can lead to conclusions about well-defined regions and prediction reliability.

Instead of an infeasible examination of the suboptimal structure space, we want to answer the following question: Given an RNA sequence $S$ with the set of all possible RNA structures $\mathcal{R} = \{R_1, \ldots, R_m\}$, what is the probability $p_i$ for the formation of structure $R_i$ with free energy $E(R_i)$? In particular, we are interested in the probability distribution $\overrightarrow{p} = (p_1, \ldots, p_m)$. There is no knowledge about the distribution except that there are $m$ states. When estimating a probability distribution without any additional information, we want to choose the one with maximum uncertainty (*i.e.*, *maximum entropy*). This returns the most unbiased probability distribution. The level of uncertainty is expressed by the following famous Shannon entropy formula:

$$H\left(\vec{p}\right) = -\sum_i p_i \log p_i \tag{24.11}$$

The uniform probability distribution $p_i = \frac{1}{m}$ maximizes the entropy under the side condition

$$\sum_i p_i = 1 \tag{24.12}$$

For the set of all possible RNA structures $\mathcal{R} = \{R_1, \ldots, R_m\}$ with probability distribution $\vec{p} = (p_1, \ldots, p_m)$, there is one additional constraint we can include. The expected value for the energy at equilibrium is:

$$< E >= \sum_i p_i E(R_i) = \text{constant} \tag{24.13}$$

Taking the side condition and this additional constraint, we get the maximum entropy for the following probability distribution:

$$p_i = \frac{e^{-E(R_i)/kT}}{Q} \tag{24.14}$$

This probability distribution is referred to as the *Boltzmann distribution* with Boltzmann constant $k$ and temperature $T$. $Q$ is the *partition function*, defined as the weighted sum over the set of all possible structures $\mathcal{R} = \{R_1, \ldots, R_m\}$:

$$Q = \sum_i e^{-E(R_i)/kT} \tag{24.15}$$

The probability of a given structure $R_i$ with free energy $E(R_i)$ is proportional to $e^{-E(R_i)/kT}$ in equilibrium at temperature $T$. To calculate explicitly the probability of a structure $R_i$ for a set of possible structures, the partition function $Q$ needs to be known. In general, partition function calculation using direct enumeration over all possible states of the system is infeasible. However, for RNA structures, the partition function can be determined explicitly. Here, the formula for calculation of energy $E(R_i)$ for structure $R_i$ in the nearest-neighbor energy model is used (Section 24.2.1).

$$E(R_i) = \sum_{(i,j)\in R_i} E_{i,j}^{R_i} \tag{24.16}$$

With these additive terms, one can rewrite the partition function calculation as follows:

$$Q = \sum_i e^{-E(R_i)/kT} = \sum_i \prod_{(i,j)\in R_i} e^{-E^{R_i}_{i,j}/kT} \tag{24.17}$$

The number of terms increases exponentially with the sequence length. The key concept for efficient partition function calculation is a restructuring of terms so that dynamic programming can be applied. The same principle as in MFE folding is used (*i.e.*, the additive energy function based on the set of secondary structure elements [44]). However, the recursion scheme differs because it is well known that there are redundancies in the dynamic programming algorithm. For the partition function calculation, each structure element must be handled exactly once. If we count the same state more than once then the partition function will be wrong. An elegant and efficient way for computing the full equilibrium partition function for noncrossing RNA secondary structure was introduced in [27]. The algorithmic details are described as follows:

**Input:** RNA sequence $S = S_1 \ldots S_n$ with $S_i \in \{A, C, G, U\}$.
**Output:** Full equilibrium partition function $Q$ for $S$ defined as the Boltzmann weighted free energy sum over the set of all possible secondary structure elements. From the partition function $Q$, probabilities for structure elements and base pairs can be derived.

Following the dynamic programming principle, the algorithm starts with subsequences of length one and continues with subsequences of increasing lengths. The solution for a subsequence is assembled from already computed solutions for smaller subsequences. The main consideration is that every state must be handled exactly once. Therefore, we need to find a slightly different recursion scheme for calculating the partition function $Q$ as the one used in MFE folding. The full equilibrium partition function $Q$ over subsequence $S = S_i \ldots S_j$ can be expressed through the following equation:

$$Q_{i,j} = 1 + \sum_{\substack{k,l,\\ i\le k<l\le j}} Q_{i,k-1} Q^{\mathrm{b}}_{k,l} \tag{24.18}$$

where, $Q^{\mathrm{b}}_{i,j}$ represents the partition function over subsequence $S = S_i \ldots S_j$ with the requirement that $(i, j)$ is a valid base pair. Decomposition of structures according to the last base pair guarantees that no structure element is counted twice in the Boltzmann weighted free energy sum. The recursions for $Q^{\mathrm{b}}_{i,j}$ must cover all possible structure elements (helix, hairpin loop, bulge and internal loop, and multiloop). For the detailed equations, we refer the reader to the original papers [27] and [11]. A visualization of the matrices is shown in Figure 24.7 following the diagrams used in Section 24.2.2 [11]. Overall, the partition function calculation using dynamic programming takes $O(n^4)$ time and $O(n^2)$ space and considers all possible secondary

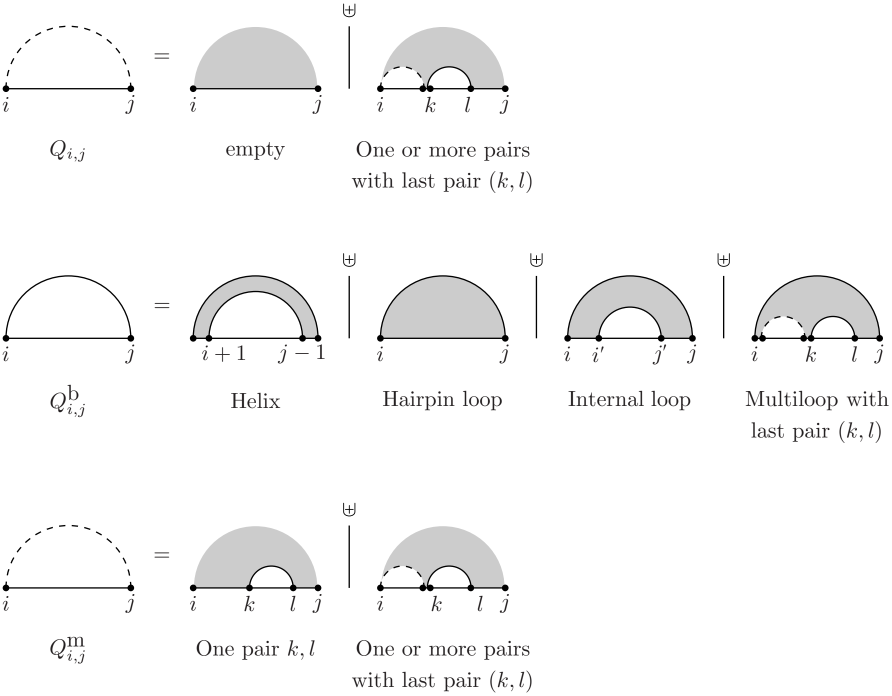


**Figure 24.7** Recursion scheme for the matrices used in the dynamic programming algorithm for partition function calculation. Matrix $Q^{\mathrm{m}}_{i,j}$ describes the multiloop calculation.

structures. With certain internal loop restrictions as in MFE folding, run time can be reduced to $O(n^3)$ [27, 25, 11].

### 24.2.4 Base Pair Probabilities

The partition function calculation for RNA structures is a powerful tool for folding space statistics in equilibrium. In contrast to prediction methods that return a structure with minimum free energy (and only few suboptimal solutions within a certain energy range), the partition function considers the ensemble of all possible structures. As introduced earlier, the equilibrium probability of a given structure $R_i$ with free energy $E(R_i)$ is proportional to $e^{-E(R_i)/kT}$ at temperature $T$. Furthermore, one can derive probabilities for substructures from the partition function, in particular base pair probabilities.

Given base pair $(i, j)$, its probability $p_{(i,j)}$ is defined as the sum of probabilities for all structures $R_l \in \mathcal{R}$ that contain base pair $(i, j)$:

$$p_{(i,j)} = \sum_{R_l \ni (i,j)} \frac{e^{-E(R_l)/kT}}{Q} \tag{24.19}$$

It is important to note that base pair probabilities are not independent. Therefore, it is wrong simply to multiply base pair probabilities to compute the probability of a structure element. The probability for a helix, hairpin loop, bulge loop, internal loop, or multiloop must be calculated using the partition function.

Base pair probabilities also have been used to determine whether a structured region is well defined [22]. The most robust measure is the positional entropy $S_k$ for a base $k$:

$$S_k = -\sum_i p_{(i,k)} \log p_{(i,k)} \tag{24.20}$$

where, $p_{(i,i)}$ is the probability that base $i$ does not form a pair with any other base:

$$p_{(i,i)} = 1 - \sum_{j \neq i} p_{(i,j)} \tag{24.21}$$

Regions with a high positional entropy form many alternative structures and are not well defined. Positional entropy values can be used to assess the reliability of a predicted secondary structure.

Base pair probabilities $p_{(i,j)}$ for a sequence $S$ conveniently are represented in a symmetric $n \times n$ matrix. The entry $(i, j)$ contains the probability that bases $i$ and $j$ form a base pair. For visualization, a so-called *probability dot plot* is common. The probability dot plot is a concise representation of the ensemble folding statistics. In dot plots, the upper triangle displays base pair probabilities, whereas the lower triangle contains a square for each base pair which belongs to the minimum free energy structure (Figure 24.8). In the upper triangle, the size of the squares is proportional to the base pair probabilities. The tool `RNAfold` from the Vienna RNA Package produces such a base pair probability dot plot in a PostScript file [20].

### ■ EXAMPLE 24.2

The Vienna `RNAfold` web server was used for the structure prediction of an example transfer RNA (tRNA) [20]. The results are shown in Figure 24.8. The structure is returned in dot bracket notation, which represents unpaired and paired positions. The probability dot plot displays base pair probabilities in the upper right triangle and the minimum free energy structure in the lower left triangle. Alternatively, one can use the famous `mfold` web server for minimum free energy folding [43]. It supports the prediction of suboptimal structures, however, without the use of a partition function calculation.

## 24.3 RNA PSEUDOKNOTS

Things become intricate when crossing structure elements come into play. A structure $R$ is called *crossing* or *overlapping* if it contains at least two base pairs

```
GGGCUAUUAGCUCAGUUGGUUAGAGCGCACCCCUGAUAAGGGUGAGGUCGCUGAUUCGAAUUCAGCAUAGCCCA
(((((((..((((.........)))).(((((.......))))).....(((((.......))))))))))))).
```

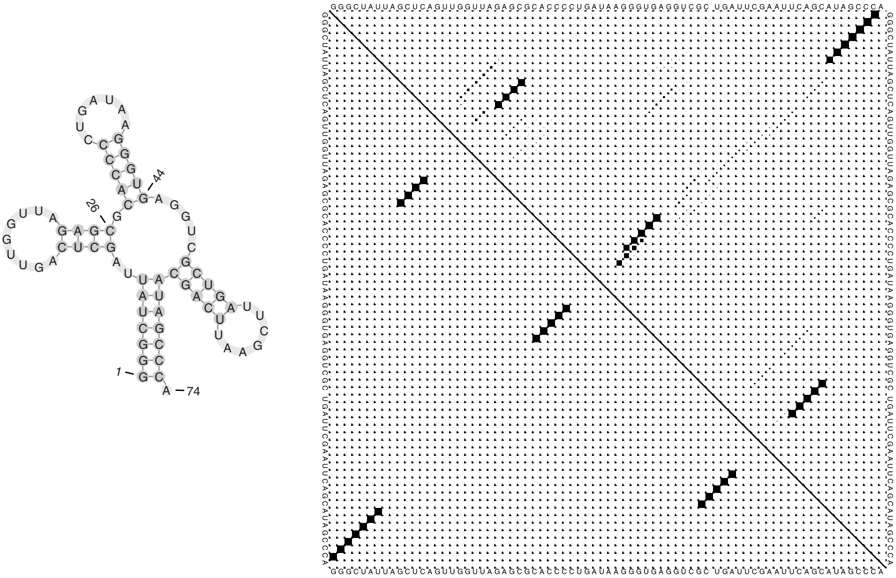

26
44
1
A −74
GGGCUAUUAGCUCAGUUGGUUAGAGCGCACCCCUGAUAAGGGUGAGGUCGC UGAUUCGAAUUCAGCAUAGCCCA

**Figure 24.8** An example structure prediction for a tRNA sequence. The dot bracket notation is shown at the top, and the secondary structure is visualized in the lower left using the PseudoViewer software [19]. The probability dot plot is shown on the right and can be used to judge the reliability of the predicted minimum free energy structure.

$(i, j), (k, l) \in R$ with $i < k$, which fulfill $i < k < j < l$. The notion of crossing base pairs is important as these may result in the formation of a functional structure element, a so-called *pseudoknot* (Figure 24.9). In RNA structures, pseudoknots form when bases within a loop region pair with complementary unpaired bases outside the loop. This results in two pseudoknot helices or stems $S_1$ and $S_2$ and three pseudoknot loops $L_1$, $L_2$, and $L_3$. It is important to note that the two pseudoknot loops $L_1$ and $L_3$ are not equivalent from a thermodynamic viewpoint. Loop $L_1$ spans across the major groove of stem $S_2$, and loop $L_3$ spans across the minor groove of stem $S_1$ (Figure 24.9b). Therefore, it is safe to assume that the loop entropies for $L_1$ and $L_3$ are not equivalent stereochemically. It is still an open question how to model the crucial stem-loop correlations as there is little experimental data on pseudoknot thermodynamics, and progress only has been made recently [6, 7, 8].

Formally, pseudoknots can be described as secondary structures (*i.e.*, a set of crossing and noncrossing base pairs). However, they often are attributed to tertiary structure in the literature because of the hierarchical two-step process assumed to aid in their formation [39]. In the following sections, we will introduce pseudoknots in more detail and describe practical prediction methods from the literature.

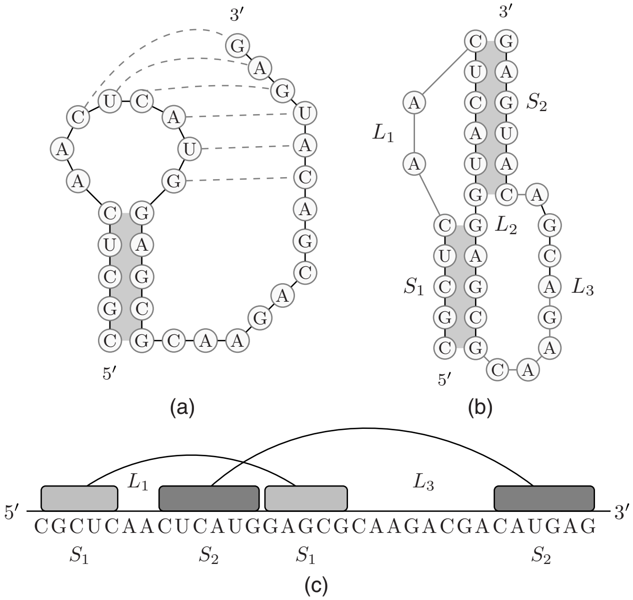


**Figure 24.9** Structure of a pseudoknot. (a) Unpaired bases in a hairpin loop bond with complementary unpaired bases outside the loop. (b) The resulting pseudoknot has two stems and three loops. Loop $L_2$ at the junction of the two stems has zero nucleotides in this example. This can lead to extra stability resulting from coaxial stacking of the stems. (c) The pseudoknot can be drawn as crossing intervals on the line.

## 24.3.1 Biological Relevance

Pseudoknots first were identified in 1982 but received only little interest over the following decade [31]. However, in the last couple of years, numerous pseudoknot features and roles started to unravel. Recent studies on RNA emphasize the fact that pseudoknots are essential structures in many biological processes and viruses [3, 16]. A few selected examples will be given here, for two excellent reviews on the diversity of pseudoknots; see [36] and [4].

Many pseudoknots are found in RNA viruses where they exhibit various functions. They are of high medical research interest, as many viruses use them for infecting the host cell. For example, pseudoknots have been found to be involved in translation initiation at viral and cellular internal ribosome entry sites (IRESs) [3]. IRES is a highly structured stretch of RNA sequence located in the 5′-untranslated region (UTR), which mimics the capped structure for ribosome recruitment. Therefore, most protein factors required for conventional translation initiation no longer are needed, which is a particular advantage in a competitive host cell environment.

After translation initiation, reprogramming can take place on the mRNA level. During translation, a ribosome is forced to switch to one of the alternative reading frames. Especially, viruses use programmed –1 ribosomal frameshift to produce more than one protein from overlapping open reading frames in a single mRNA [16]. RNA frameshifting pseudoknots have been studied extensively because of the impact of devastating viruses (*e.g.*, severe acute respiratory syndrome (SARS)) and some nuclear magnetic resonance (NMR) solution structures are available [17].

In positive-strand RNA viruses, the 3′-UTR can form a tRNA-like structure (TLS) instead of a standard mRNA poly(A) tail. Coaxially stacked pseudoknots mimic a tRNA cloverleaf structure. These pseudoknots are assumed to be involved in translation and viral genome replication [12].

Another class of pseudoknots has been discovered in a multifunctional type of RNA: bacterial transfer-messenger RNA (tmRNA). Ribosomes can stall during protein synthesis as a result of mRNAs without a stop codon. Dual tRNA and mRNA, so-called tmRNA, perform a rescue process called trans-translation. The broken mRNA is liberated from the paused ribosome, and the incomplete polypeptide gets marked as a degradation target. In general, bacterial tmRNAs consist of a tRNA-like structure, a mRNA-like structure, and four pseudoknots [16].

Pseudoknots also have been found in catalytically active RNAs, for example, in ribozymes. Some ribozymes comprise long-range interactions, double pseudoknots, or kissing interactions to support a complex three-dimensional folding for catalytic activity. One well-studied example is the hepatitis delta virus (HDV) ribozyme. The HDV is a human pathogen causing severe illness. The HDV genome features a catalytic component for self-cleaving. This ribozyme is essential for viral replication, as it cuts RNA transcripts into unit lengths. The 72-nucleotide genomic HDV ribozyme is the fastest known naturally occurring self-cleaving RNA and features a nested double pseudoknot configuration [13].

### 24.3.2 RNA Pseudoknot Prediction

Pseudoknots commonly occur in RNA and perform essential functions as part of cellular transcription machinery, regulatory processes, and viral replication. Therefore, including pseudoknots in structure prediction is relevant for a deeper understanding of RNA functions and moreover for antiviral drug design. Contrary to their biological importance, the vast majority of RNA structure prediction methods simply exclude pseudoknots altogether. Dynamic programming algorithms for RNA secondary structure prediction cannot accommodate pseudoknots. Recall that in a pseudoknot, there will be at least one crossing base pair that violates the assumption of autonomous subproblems. From a computer scientists point of view, including a crossing structure element like the pseudoknot dampens the optimism of solving the RNA structure prediction problem. It has been shown that the general prediction of pseudoknots under a simple energy model is a nondeterministic polynomial (NP)-complete problem [24]. Therefore, an exact polynomial time algorithm under the (even more complicated) minimum free energy model does not exist (unless P = NP). Three routes in the literature can be taken to include pseudoknots in structure prediction (Figure 24.10), and they are:

- Complex dynamic programming for the prediction of structures including restricted sorts of pseudoknots
- Heuristic algorithms for the prediction of structures including pseudoknots
- Pseudoknot detection for finding pseudoknots in a sequence

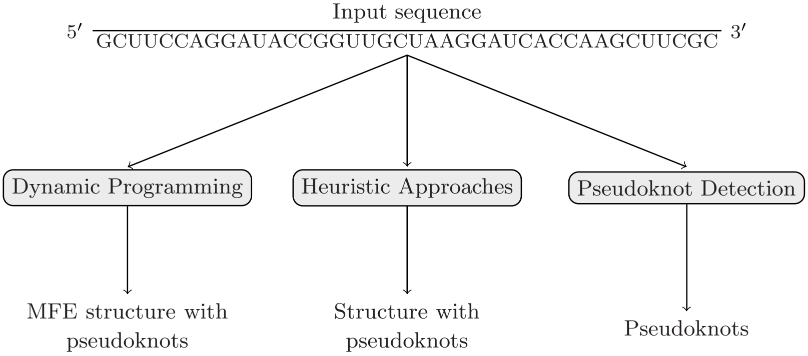

Input sequence
5′ GCUUCCAGGAUACCGGUUGCUAAGGAUCACCAAGCUUCGC 3′
Dynamic Programming
Heuristic Approaches
Pseudoknot Detection
MFE structure with pseudoknots
Structure with pseudoknots
Pseudoknots

**Figure 24.10** Three main approaches in RNA pseudoknot prediction.

Another popular approach for RNA structure prediction are kinetic folding simulations. However, we will not elaborate on the details and rather refer the reader to reviews as in [14, 34].

### 24.3.3 Dynamic Programming

Dynamic programming algorithms for pseudoknot prediction need to employ a restricted class of pseudoknots to achieve reasonable run time. However, the pseudoknot classes that are predicted by dynamic programming often cannot be predefined clearly. They only are given implicitly through the complex recursion scheme and therefore remain ambiguous in many cases. Here, we present several dynamic programming algorithms and sketch the corresponding pseudoknot target class. A hierarchy of pseudoknot classes will be described, according to [10].

***24.3.3.1 Rivas and Eddy (1999).*** The dynamic programming algorithm `pknots` comprises the most general class of pseudoknots [32]. We will not present the recursion scheme in detail because it consists of many more matrices and recursion equations than the algorithm presented in Section 24.2.2. The main idea is the use of gap matrices to describe pseudoknots (for details, see [32]). Gap matrices are the foundation for the overall recursion scheme, which is complex and powerful. However, the exact class of pseudoknot configurations that can be folded remains elusive. The algorithm can predict some nonplanar configurations, such as the two pseudoknots shown in Figure 24.11. On the other hand, not all nonplanar pseudoknots are covered through the recursion scheme (*e.g.*, the group II intron [10]). There are also some planar pseudoknots that `pknots` cannot predict [24]. An implementation is available that runs in $O(n^6)$ time and $O(n^4)$ space and is only feasible for sequences shorter than 150 nucleotides.

***24.3.3.2 Akutsu (2000).*** This dynamic programming algorithm for the more restricted class of *simple recursive pseudoknots* demands $O(n^5)$ time and $O(n^3)$

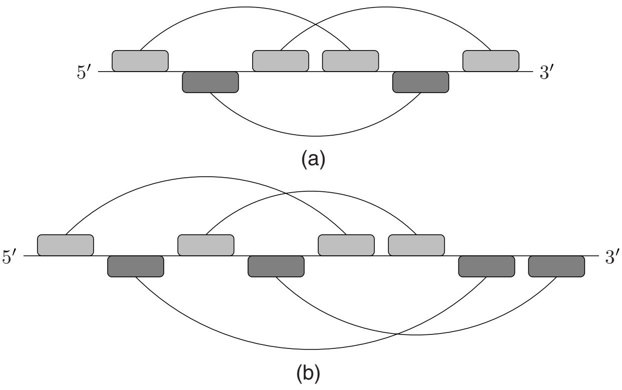


**Figure 24.11** Two examples of nonplanar pseudoknots where (b) is the double pseudoknot structure for the $\alpha$ operon mRNA in *Escherichia coli* [38]. A nonplanar pseudoknot cannot be drawn in the plane without intersections of arcs.

space [2]. The class of simple recursive pseudoknots is contained fully in the pknots recursion scheme and therefore is a proper subset [24]. A *simple pseudoknot* in an RNA sequence $S$ forms if there are positions $S_{i_0}, S_{j'_0}, S_{j_0}, S_{k_0}$ with $i_0 < j'_0 < j_0 < k_0$ for which the following holds:

- Each participating base pair $(i, j)$ satisfies either $i_0 \leq i < j'_0 \leq j < j_0$ or $j'_0 \leq i < j_0 \leq j \leq k_0$.
- If participating base pairs $(i, j)$ and $(i', j')$ satisfy either $i < i' < j'_0$ or $j'_0 \leq i < i'$, then $j' < j$.

This class can be extended to *simple recursive pseudoknots*. Here, further secondary structure elements or pseudoknots are allowed within the loop regions. It is clear that simple (recursive) pseudoknots belong to the class of planar pseudoknots (Figure 24.12). However, they are only a subset as there are planar pseudoknots that are

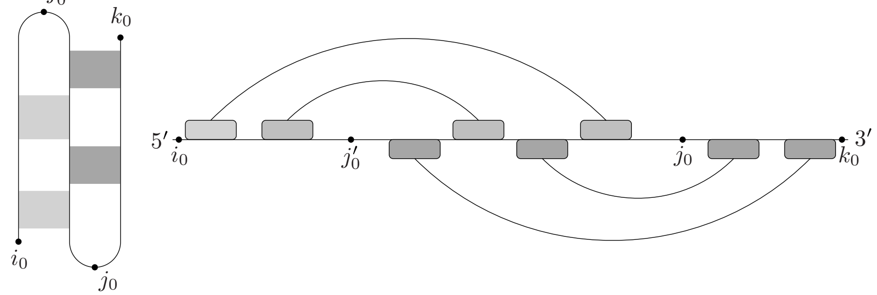


**Figure 24.12** Structure of a simple pseudoknot. The interval representation shows that the base pairs in the upper (or lower) half are not allowed to cross; therefore, simple (recursive) pseudoknots are a subset of planar pseudoknots.

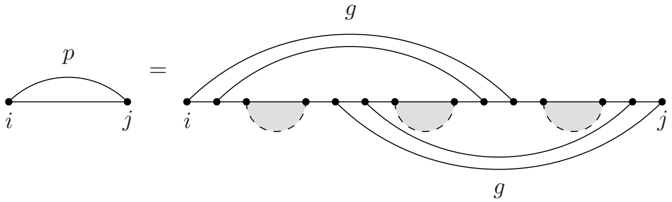


**Figure 24.13** Pseudoknot recursion scheme used in NUPACK. A pseudoknot *p* for subsequence $S_i \ldots S_j$ is constructed with two gap matrices *g*. Recursive internal structures or pseudoknots are allowed in the loops.

not included in the definition of simple recursive pseudoknots [24]. One should note that under a very simple energy model, the computation time of this algorithm can be reduced to $O(n^4)$ [2].

***24.3.3.3 Dirks and Pierce (2003).*** The dynamic programming algorithm NUPACK can predict a more restricted class of pseudoknots, which is defined implicitly through the recursion scheme [11]. It takes $O(n^5)$ time and $O(n^4)$ space. One should note that the high space requirements stem from the included partition function calculation. Basically, a pseudoknot is constructed through two gap matrices with possible recursive structures (Figure 24.13). Gap matrices may contain interior loops or even multiloops. This class is a proper subset of the simple recursive pseudoknots defined by Akutsu [2] and therefore is a proper subset of the class by Rivas and Eddy [32].

***24.3.3.4 Lyngsø and Pedersen (2000).*** In this work, a restricted class of pseudoknots is proposed [23]. The algorithm handles only a subset of the *simple recursive pseudoknots* defined by [2]; however, it has the same time and space requirements. Here, a pseudoknot consist of four parts, where opposing subsequences are allowed to form pseudoknot-free secondary structures. This class of pseudoknots is described

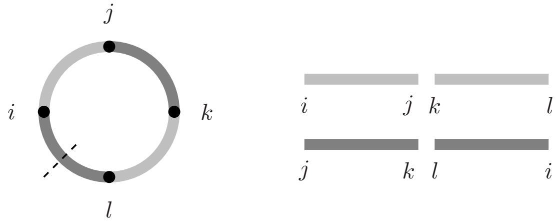


**Figure 24.14** An abstract model for a class of pseudoknots is shown [23]. Here, the sequence is divided into four parts. $S_i \ldots S_j$ and $S_k \ldots S_l$ ($S_j \ldots S_k$ and $S_l \ldots S_i$) are called opposing subsequences. The opposing subsequences are allowed to form arbitrary secondary structure elements. The crossing interaction for the pseudoknot is achieved through a combination of the four parts. Note that the sequence is drawn as a circle, and one of the four parts can extend over the sequence end (shown as black line).

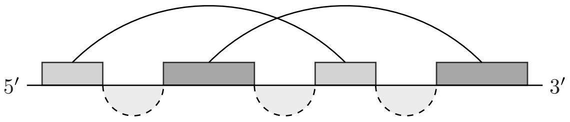


**Figure 24.15** Canonical simple recursive pseudoknots with two crossing stems and possible secondary structure elements or pseudoknots in the three loop regions. Canonical simple recursive pseudoknots are a proper subset of the class described in [11].

in a rather abstract manner (Fig. 24.14). It may not be obvious from first sight, but it includes kissing hairpins; however, recursive pseudoknots are not allowed.

***24.3.3.5 Reeder and Giegerich (2004).*** The predefined class of *canonical simple recursive pseudoknots* is the target for pknotsRG, a dynamic programming algorithm, which runs in $O(n^4)$ time using $O(n^2)$ space [29]. A simple recursive pseudoknot is defined as in [2] and allows internal secondary structures or pseudoknot formation in each of the three loop regions (Figure 24.15). Note that base pairing between different loops is not allowed. Furthermore, the following canonization rules are introduced:

1. No bulges or internal loops are allowed in the stems. Therefore, both strands in a stem must have the same length.[1]
2. The two crossing pseudoknot stems must have the maximum number of possible base pairs.
3. If the two crossing, maximal pseudoknot stems compete for a base pair, then the boundary is fixed at an arbitrary point between them.

### 24.3.4 Heuristic Approaches

Apart from dynamic programming, heuristic approaches have been developed for RNA structure prediction including pseudoknots. Early methods comprise Monte Carlo simulations [1], genetic algorithms [18, 40], stochastic context-free grammars [5], and maximum weighted matching (MWM) based on graph theory [37]. In a folding graph, vertices correspond to bases and edge weights to base pair scores. The folding with overall maximum edge weights can be found in $O(n^3)$ time and $O(n^2)$ space. As the underlying energy model is very simple and neglects loop entropies, prediction accuracy strongly depends on the base pair score reliability. With a meaningful alignment as a prerequisite, MWM leads to good results. However, the method has low prediction accuracy for *ab initio* structure prediction. Elaborated stochastic folding simulations are performed in KineFold [41]. Run time is relatively high; therefore, this kinetic approach is restricted to short sequences.

Recent heuristic methods attempt to model the RNA folding pathway by iteratively adding promising stems from a candidate pool to the structure [33, 30, 9]. The

[1] Note that this rule has been relaxed to accommodate bulge loops with one nucleotide [29].

final result can contain pseudoknots of any type. However, there are a few pitfalls regarding this approach. First, most commonly, the candidate stems are selected using simple base pair maximization. Such a procedure only takes into account the local stabilizing forces of base pairing, yet no loop entropies or interaction of stems. Second, a greedy stem adding procedure often fails in practice.

### 24.3.5 Pseudoknot Detection

A different route is followed in pseudoknot search programs. Here, one attempts to identify possible pseudoknot candidates in a sequence as a first step. From this candidate pool, sequence fragments can be analyzed regarding their pseudoknot folding potential. Pseudoknot detection is computationally efficient and easily can incorporate more sophisticated energy rules for pseudoknots. Detection of pseudoknots must be distinguished clearly from RNA structure prediction including pseudoknots. Pseudoknot detection is a self-contained step without simultaneous secondary structure prediction aimed to return only pseudoknots. If pseudoknots can be detected with high accuracy, then the remaining sequence can be folded efficiently using state-of-the-art secondary structure prediction programs in $O(n^3)$ time and $O(n^2)$ space.

HPknotter is such a detection tool for pseudoknots based on structural matching and dynamic programming verification [21]. It relies on the observation that if presented with a sequence fragment exactly harboring a pseudoknot, then dynamic programming methods can fold it into the correct structure with high base pair accuracy. HPknotter can improve RNA secondary structure prediction including pseudoknots. However, it suffers from a high number of returned false positive pseudoknots. A more refined heuristic pseudoknot detection tool is KnotSeeker [35]. It uses a hybrid sequence matching and free energy minimization approach to perform a screening of the primary sequence. Short sequence fragments are selected as possible candidates that may contain pseudoknots and are verified using an existing dynamic programming algorithm and a minimum weight independent set calculation.

### 24.3.6 Overview

Dynamic programming algorithms for RNA structure prediction including pseudoknots have been studied in detail, and several methods exist (Table 24.1). Practical

**Table 24.1 Dynamic programming algorithms for pseudoknot prediction**

| Reference | Time | Space | Pseudoknot Class | Implementation |
|---|---|---|---|---|
| Rivas and Eddy (1999) | $O(n^6)$ | $O(n^4)$ | Broad | `pknots` |
| Akutsu (2000) | $O(n^5)$ | $O(n^3)$ | Simple recursive | — |
| Lyngsø and Pedersen (2000) | $O(n^5)$ | $O(n^3)$ | Restricted simple recursive | — |
| Dirks and Pierce (2003) | $O(n^5)$ | $O(n^4)$ | Restricted simple recursive | `NUPACK` |
| Reeder and Giegerich (2004) | $O(n^4)$ | $O(n^2)$ | Canonical simple recursive | `pknotsRG` |

**Table 24.2 Selected heuristic methods for pseudoknot prediction and detection**

| Name | Description |
|---|---|
| MWM | Comparative or *ab initio* matching in folding graph |
| ILM | Comparative or *ab initio* iterative stem adding procedure |
| HotKnots | Iterative stem adding procedure with suboptimal scenarios |
| HPknotter | Pseudoknot search tool based on structural matching |
| KnotSeeker | Pseudoknot search tool based on RNA folding |

run time only can be achieved by restricting the class of predictable pseudoknots or by employing a very simple energy model (such as base pair maximization). However, too many restrictions are not desirable, and therefore, most algorithms are computationally very expensive. The main advantage of dynamic programming algorithms is their consistence with the minimum free energy model for secondary structure elements. However, for pseudoknots, they typically employ an oversimplified energy model. This is done for two reasons. First, not many experiments have been done on pseudoknot thermodynamics and stem-loop correlations [8]. Second, a simplified pseudoknot energy model often is easier to integrate in the dynamic programming framework. Similar to multiloops, the free energy of a pseudoknot is estimated by an initiation penalty, the stabilizing helix free energies, and a penalty for each unpaired nucleotide in the loops. This approximation can lead to poor results, especially for longer sequences.

Heuristic methods for pseudoknot prediction are generally more efficient than dynamic programming (Table 24.2). A major disadvantage is that unlike dynamic programming, there is generally no guarantee of finding a solution with minimum free energy. On the other hand, heuristic methods tend to have a flexible framework and better efficiency than dynamic programming. For example, some methods can include comparative information [37, 33]. As a concluding remark, most heuristic methods simply have adopted the oversimplified pseudoknot energy model also used in dynamic programming. In practice, this is a major disadvantage as it contributes to poor prediction results for longer sequences.

Heuristic pseudoknot detection is a promising approach still under development (Table 24.2). The detection of pseudoknots is very fast and practical, and therefore is suited for structure prediction of long RNA sequences. The algorithmic framework allows easy integration of sophisticated energy rules during pseudoknot candidate verification. There is no guarantee of finding the overall minimum free energy structure. However, after pseudoknot detection, the remaining sequence can be folded using state-of-the-art MFE prediction algorithms.

## 24.4 CONCLUSIONS

Computational RNA structure prediction has come a long way in the last 20 years. We gave a brief overview of the fundamental dynamic programming algorithm for RNA structure prediction. Current trends in the literature are an improvement of the energy model, examination of the suboptimal folding space, and the inclusion of pseudoknots in algorithmic predictions. It is behind the scope of this chapter to cover all aspects in depth; therefore, emphasis was laid on pseudoknots because of their biological relevance. This chapter will benefit computer scientists looking for an introduction to RNA structure prediction with pseudoknots. Last, but not least, we hope to give biologists insight into what is happening behind the scences of the prediction tools they use.

## REFERENCES

1. J.P. Abrahams, M. van den Berg, E. van Batenburg, and C. Pleij. Prediction of RNA secondary structure, including pseudoknotting, by computer simulation. *Nucleic Acids Res*, 18(10):3035–3044, 1990.
2. T. Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Appl Math*, 104(1-3):45–62, 2000.
3. I. Brierley, R.J. Gilbert, and S. Pennell. RNA pseudoknots and the regulation of protein synthesis. *Biochem Soc Trans*, 36(4):684–689, 2008.
4. I. Brierley, S. Pennell, and R.J. Gilbert. Viral RNA pseudoknots: versatile motifs in gene expression and replication. *Nat Rev Microbiol*, 5(8):598–610, 2007.
5. L. Cai, R.L. Malmberg, and Y. Wu. Stochastic modeling of RNA pseudoknotted structures: a grammatical approach. *Bioinformatics*, 19:i66–i73, 2003.
6. S. Cao and S.J. Chen. Predicting RNA pseudoknot folding thermodynamics. *Nucleic Acids Res*, 34(9):2634–2652, 2006.
7. S. Cao and S.J. Chen. Predicting structures and stabilities for H-type pseudoknots with interhelix loops. *RNA*, 15(4):696–706, 2009.
8. S.J. Chen. RNA folding: Conformational statistics, folding kinetics, and ion electrostatics. *Annu Rev Biophys*, 37:197–214, 2008.
9. X. Chen, S. He, D. Bu, F. Zhang, Z. Wang, R. Chen, and W. Gao. FlexStem: Improving predictions of RNA secondary structures with pseudoknots by reducing the search space. *Bioinformatics*, 24(18):1994–2001, 2008.
10. A. Condon, B. Davy, B. Rastegari, S. Zhao, and F. Tarrant. Classifying RNA pseudoknotted structures. *Theor Comput Sci*, 320(1):35–50, 2004.
11. R.M. Dirks and N.A. Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *J Comput Chem*, 24(13):1664–1677, 2003.
12. T.W. Dreher and W.A. Miller. Translational control in positive strand RNA plant viruses. *Virology*, 344(1):185–197, 2006.
13. A.R. Ferre-D’Amare, K.H. Zhou, and J.A. Doudna. Crystal structure of a hepatitis delta virus ribozyme. *Nature*, 395(6702):567–574, 1998.
14. C. Flamm and I.L. Hofacker. Beyond energy minimization: approaches to the kinetic folding of RNA. *Monatsh. Chem.*, 139(4):447–457, 2008.
15. P.P. Gardner and R. Giegerich. A comprehensive comparison of comparative RNA structure prediction approaches. *BMC Bioinformatics*, 5(140):2004.
16. D.P. Giedroc and P.V. Cornish. Frameshifting RNA pseudoknots: Structure and mechanism. *Virus Res*, 139(2):193–208, 2009.

17. D.P. Giedroc, C.A. Theimer, and P.L. Nixon. Structure, stability and function of RNA pseudoknots involved in stimulating ribosomal frameshifting. *J Mol Biol*, 298(2):167–185, 2000.

18. A.P. Gultyaev, F.H. van Batenburg, and C.W. Pleij. The computer simulation of RNA folding pathways using a genetic algorithm. *J Mol Biol*, 250(1):37–51, 1995.

19. K. Han and Y. Byun. PSEUDOVIEWER2: Visualization of RNA pseudoknots of any type. *Nucleic Acids Res*, 31(13):3432–3440, 2003.

20. I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatsh Chem*, 125(2):167–188, 1994.

21. C.H. Huang, C.L. Lu, and H.T. Chiu. A heuristic approach for detecting RNA H-type pseudoknots. *Bioinformatics*, 21(17):3501–3508, 2005.

22. M. Huynen, R. Gutell, and D. Konings. Assessing the reliability of RNA folding using statistical mechanics. *J Mol Biol*, 267(5):1104–1112, 1997.

23. R.B. Lyngsø and C.N. Pedersen. Pseudoknots in RNA secondary structures. *Proceedings of the 4th Annual International Conference on Computational Molecular Biology*, 2000, pp. 201–209.

24. R.B. Lyngsø and C.N. Pedersen. RNA pseudoknot prediction in energy-based models. *J Comput Biol*, 7(3-4):409–427, 2000.

25. R.B. Lyngsø, M. Zuker, and C.N.S. Pedersen. Fast evaluation of internal loops in RNA secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999.

26. D.H. Mathews, J. Sabina, M. Zuker, and D.H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J Mol Biol*, 288(5):911–940, 1999.

27. J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

28. R. Nussinov and A.B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proc Natl Acad Sci U S A*, 77(11):6309–6313, 1980.

29. J. Reeder and R. Giegerich. Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, 5:104, 2004.

30. J. Ren, B. Rastegari, A. Condon, and H.H. Hoos. HotKnots: Heuristic prediction of RNA secondary structures including pseudoknots. *RNA*, 11(10):1494–1504, 2005.

31. K. Rietveld, R. Van Poelgeest, C.W. Pleij, J.H. Van Boom, and L. Bosch. The tRNA-like structure at the 3′ terminus of turnip yellow mosaic virus RNA. Differences and similarities with canonical tRNA. *Nucleic Acids Res*, 10(6):1929–1946, 1982.

32. E. Rivas and S.R. Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J Mol Biol*, 285(5):2053–2068, 1999.

33. J. Ruan, G.D. Stormo, and W. Zhang. An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. *Bioinformatics*, 20(1):58–66, 2004.

34. P. Schuster. Prediction of RNA secondary structures: From theory to models and real molecules. *Rep Prog Phys*, 69(5):1419–1477, 2006.

35. J. Sperschneider and A. Datta. KnotSeeker: Heuristic pseudoknot detection in long RNA sequences. *RNA*, 14(4):630–640, 2008.

36. D.W. Staple and S.E. Butcher. Pseudoknots: RNA structures with diverse functions. *PLoS Biol*, 3(6):e213, 2005.

37. J.E. Tabaska, R.B. Cary, H.N. Gabow, and G.D. Stormo. An RNA folding method capable of identifying pseudoknots and base triples. *Bioinformatics*, 14(8):691–699, 1998.

38. C.K. Tang and D.E. Draper. Unusual mRNA pseudoknot structure is recognized by a protein translational repressor. *Cell*, 57(4):531–536, 1989.

39. I. Tinoco and C. Bustamante. How RNA folds. *J Mol Biol*, 293(2):271–281, 1999.

40. F.H. van Batenburg, A.P. Gultyaev, and C.W. Pleij. An APL-programmed genetic algorithm for the prediction of RNA secondary structure. *J Theor Biol*, 174(3):269–280, 1995.

41. A. Xayaphoummine, T. Bucher, F. Thalmann, and H. Isambert. Prediction and statistics of pseudoknots in RNA structures using exactly clustered stochastic simulations. *Proc Natl Acad Sci U S A*, 100(26):15310–15315, 2003.

42. T.B. Xia, J. SantaLucia, M.E. Burkard, R. Kierzek, S.J. Schroeder, X.Q. Jiao, C. Cox, and D.H. Turner. Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson-Crick base pairs. *Biochemistry*, 37(42):14719–14735, 1998.

43. M. Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res*, 31(13):3406–3415, 2003.

44. M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Res*, 9(1):133–148, 1981.

# IV

# PHYLOGENY RECONSTRUCTION

# 25

# PHYLOGENETIC SEARCH ALGORITHMS FOR MAXIMUM LIKELIHOOD

Alexandros Stamatakis

## 25.1 INTRODUCTION

The reconstruction of phylogenetic (evolutionary) trees from molecular sequence data is a comparatively old problem in bioinformatics, given that Joe Felsenstein's seminal paper [12] on computing the maximum likelihood (ML) score on trees already was published in 1981 (*i.e.*, almost 30 years ago). However, significant advances in wet lab molecular sequencing technologies, such as, for instance, the introduction of 454 sequencers [53], are generating a highly challenging and unprecedented molecular data flood. Although a plethora of challenging problems exist regarding the implementation, numerical stability, and parallelization of the phylogenetic likelihood function (PLF) on emerging parallel architectures, the design of new search algorithms is equally challenging and can even yield more impressive performance improvements than high-performance computing (HPC) methods.

The PLF typically consumes more than 95% of overall execution time in all state-of-the art likelihood-based phylogeny programs. Algorithmic design for phylogeny reconstruction therefore can be regarded as trying to minimize the number of invocations of the PLF while maximizing the likelihood score of the tree that is returned by the search algorithm. Ideally, algorithmic design and HPC implementations of the PLF should be conducted simultaneously because specific heuristics may require dedicated or appropriately adapted implementations and parallelizations of the PLF.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Because parallel computing is outside the scope of this chapter, I will focus entirely on the design of search algorithms and will make an attempt to separate PLF implementation issues from abstract algorithmic ideas to the largest possible extent.

As already mentioned, the driving force behind algorithmic development is the rapid molecular data accumulation. In fact, the phyloinformatics community faces a continuous struggle to keep up with the rapid speed of data accumulation and provide ever more scalable and powerful analysis tools (*i.e.*, we just try to keep pace with data accumulation).

Memory footprints of more than 50 gigabytes (Gb), just to compute the likelihood, on a single, fixed tree as well as resource requirements exceeding 2 million central processing unit (CPU) hours simply to conduct a comprehensive and thorough real-world ML analysis on a single, large, phylogenomic dataset are becoming the norm rather than the exception. Although these datasets can be analyzed already via an admittedly rather brute-force approach on supercomputers using RAxML [58, 41], algorithmic innovations represent an often more elegant solution. In addition, algorithmic solutions that require less computational resources will allow significantly more users to conduct large-scale phylogenetic analyses (*i.e.*, will enable "large-scale phylogenetic inference for the masses").

In the present chapter, I will attempt to review the underlying concepts as well as current developments, and describe some novel experimental work on search convergence criteria as well as the efficient computation of the PLF at a purely algorithmic level. I also will outline potential future developments and challenges.

This chapter is organized as follows: In Section 25.1.1, I briefly will introduce the field of phylogenetic inference and its applications to medical and biological research. In the following Section 25.2, I will outline the basic computational steps required to compute the PLF. In the subsequent Section 25.3, I will discuss two algorithmic methods to accelerate the PLF and assess their applicability to modern phylogenetic inference problems. Thereafter, I will outline the impact of distinct alignment shapes on the design of future search algorithms (Section 25.4). In Section 25.5, I will summarize the general search mechanisms that are used and also survey the refinements thereof in some of the currently most widely used tools for phylogenetic inference under ML. In the following Section 25.6, I will outline data structures and algorithms for efficient computation of the Robinson-Foulds topological distance [47] that are a prerequisite for building the novel ML search convergence criterion that is described in the subsequent Section 25.7. I will conclude in Section 25.8 with an overview of future algorithmic challenges for the design of ML-based search algorithms.

### 25.1.1 Phylogenetic Inference

The goal of phylogenetic inference consists of reconstructing the evolutionary history of a set of $n$ present-day organisms from their respective molecular sequence data. Those $n$ organisms (also called taxa) may be represented by a concatenation of molecular data from various genes or even the whole genome. Thus, the molecular sequence representing one taxon may consist of a mixture of DNA, protein, and even morphological or binary characters. A phylogenetic tree, or simply a phylogeny, usually is represented as an unrooted binary tree, where the $n$ present-day taxa are located at the leaves of the tree, and the inner nodes represent extinct common ancestors.

The input data for a phylogenetic analysis under maximum likelihood consists of a "good" multiple sequence alignment of the $n$ taxa (*i.e.*, by insertion of gaps) or, essentially, nucleotide insertion and deletion events; all sequences in the input data will have the same length $m$ after the alignment step. Although simpler methods also exist for alignment-free tree reconstruction, they have been shown to be generally less accurate [25, 26] than alignment-based methods. Alignments that comprise sequence data from several genes are called multigene or phylogenomic alignments. A simple example for a multiple sequence alignment of DNA data for the human, the mouse, the cow, and the chicken is expressed as follows:

```
Cow      ATGGCATATCCCA-ACAACTAGGATTCCAAGA----ACATCA
Chicken  ATGGCCAACCACTCCCAACTAGGCTTTC-AGACGCCTCA-CC
Human    ATGGCACAT---GCGCAAGTAGGTCTAC-AGACGCTACT-CC
Mouse    ATGG----CCCATTCCAACTTGGTCTACAAGACGCCACATCC
```

An open issue, especially within the context of real-world analyses, is the definition of what a "good" multiple sequence alignment actually is because no objective criterion is available to judge alignment quality. In a recent *Science* paper, Loytynoja and Goldman challenged the established view of how a "good" alignment should look [33] by arguing in favor of a phylogeny-aware view of the alignment process. In addition, as tree reconstruction, the multiple sequence alignment problem is a computationally hard problem by itself. Because this chapter mainly focuses on the computational aspects of phylogenetic inference, we will just assume that the alignment is given, although the problems of phylogenetic inference and multiple sequence alignment should be solved simultaneously in an ideal world. Current approaches [75, 15, 46] for simultaneous inference of alignments and trees are still too slow and too resource-intensive for practical purposes, especially when we consider current input data growth.

It is important to note that provided any biologically meaningful optimality criterion to score a given tree topology, such as ML or maximum parsimony (MP) [14], the underlying optimization problem for finding the optimal tree is nondeterministic polynomial (NP)-hard [16, 48]. The number of distinct alternative unrooted tree topologies for $n$ taxa is $\prod_{i=3}^{n}(2i-5)$ [10]. Although a vast amount of literature exists covering heuristic search algorithms for the ML optimization problem (see [37] for an overview), they all rely on repeatedly executing the likelihood function to explore the tree space, which represents the main computational *and* memory bottleneck.

Phylogenetic trees have many important applications in medical and biological research. Current state-of-the-art ML phylogeny programs such as PAML [77], PHYML [22], PAUP* [71], GARLI [78], RAxML [58], IQPNNI [34], TREEFINDER [29], or likelihood-based Bayesian programs such as MrBayes [49], PhyloBayes [30], or BEAST [8] have accumulated more than 20,000 citations to date.

Phylogenies can be used, for example, to infer the evolutionary history of *Pappilomaviruses* that are associated with cervical cancer [20], to disentangle the evolutionary history of Acer [21] (maple trees), or to analyze bacterial communities in permafrost soils [17]. A recent phylogenomic study in *Nature* improved the accuracy of the animal tree of life [9], whereas another recent study in *Science* assessed the rates of evolution (essentially the speed of evolution) and their association to life history in flowering plants [54].

## 25.2 COMPUTING THE LIKELIHOOD

As already mentioned, the input for a phylogenetic analysis under ML consists of a multiple sequence alignment with $n$ sequences (also denoted as taxa or tips) and $m$ alignment columns. The branch length values on the tree that are returned by ML essentially represent the relative time of evolution between nodes in the tree. Here, we initially will only consider how to compute the likelihood on a fixed, given, tree topology.

Apart from the tree topology, one also requires several ML model parameters. One important parameter is the instantaneous nucleotide substitution matrix $Q$, which contains the transition probabilities for time $dt$ between nucleotide characters ($4 \times 4$ matrix, states: A, C, G, T), or for instance amino acid ($20 \times 20$ matrix) characters. The transition probability matrix for time (branch length) $t$ then is computed as $P(t) = e^{Qt}$ and can be obtained via a respective Eigenvector/Eigenvalue decomposition.

Note that, various models also exist to accommodate RNA secondary structure information (*i.e.*, models that allow to group together RNA data columns and hence let them evolve together in the respective RNA alignment). For secondary structure, six-state ($6 \times 6$ $Q$-matrix), 7-state, and 16-state models (for a summary see [52]) exist. Recently, 61-state Codon models (see, *e.g.*, [19]) for protein-coding genes, which group together triplets of DNA characters, also have received considerable attention. The computational complexity for computing the likelihood of a single column is directly proportional to the square of the number of states (*e.g.*, $O(4^2)$ for DNA data or $O(20^2)$ for protein data).

Using DNA as an example, in addition to the $Q$ matrix, we also need the prior probabilities of observing the nucleotides (*e.g.*, $\pi_A, \pi_C, \pi_G, \pi_T$) for DNA data, which either can be determined empirically from the alignment or be obtained via an ML estimate. We also need the $\alpha$ shape parameter that forms part of the $\Gamma$ model [76] of rate heterogeneity. The $\Gamma$ model accounts for the biological fact that different columns in the alignment evolve at different speeds. Although the $\Gamma$ model is well established and the de-facto standard, computationally much more efficient ways exist to incorporate rate heterogeneity, such as the CAT (rate heterogeneity with per site rate CATegories) approximation of rate heterogeneity [57]. Finally, one also requires the $2n - 3$ branch lengths in the unrooted tree topology.

Given all these parameters, to compute the likelihood of a fixed *unrooted* binary tree topology, initially one needs to compute the entries for all internal probability vectors (located at the inner nodes) that contain the probabilities $P(A)$, $P(C)$, $P(G)$, and $P(T)$ of observing an A, C, G, or T at each site/column $c$ ($c = 1 \dots m$) of the input alignment at a specific inner node. Those probability vectors are computed bottom-up from the tips toward a virtual root that can be placed into any branch of the tree. Irrespective of the placement of the virtual root, one always will obtain the same likelihood score.

This important property of ML holds if the nucleotide substitution model is time-reversible (*i.e.*, evolution occured in the same way if followed forward or backward in time). The most commonly used and general model for DNA substitution is the general time reversible (GTR) model [72] of nucleotide substitution. However, mathematical frameworks also exist for comparatively efficient and more realistic nonreversible substitution models [4]. The procedure described for computing the likelihood also is known as the Felsenstein pruning algorithm [12].

As already mentioned, every probability vector entry $\vec{L}(c)$ at position $c$ ($c = 1 \dots m$) at the tips and at the inner nodes of the tree topology, contains the four probabilities $P(A)$, $P(C)$, $P(G)$, and $P(T)$ of observing a nucleotide A, C, G, or T, at a specific column $c$ of the input alignment. The probabilities at the tips (leaves) of the tree for which observed data (*e.g.*, the DNA sequences of the currently living organisms under study) *is* available are set to 1.0 for the observed nucleotide character at the respective position $c$, (*e.g.*, for the nucleotide A: $\vec{L}(c) = [1.0, 0.0, 0.0, 0.0]$). Given a parent node $k$, and two child nodes $i$ and $j$ (with respect to the virtual root), their probability vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$, the respective branch lengths leading to the children $b_i$ and $b_j$, and the transition probability matrices $P(b_i)$, $P(b_j)$, the probability of observing an A at position $c$ of the ancestral (parent) vector $\vec{L}_A^{(k)}(c)$ is computed as follows:

$$\vec{L}_A^{(k)}(c) = \left(\sum_{S=A}^{T} P_{AS}(b_i)\vec{L}_S^{(i)}(c)\right)\left(\sum_{S=A}^{T} P_{AS}(b_j)\vec{L}_S^{(j)}(c)\right) \tag{25.1}$$

The transition probability matrix $P(b)$ for a given branch length $b$ is obtained from $Q$ via $P(b) = e^{Qb}$. Once the two probability vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$ to the left and right of the virtual root ($vr$) have been computed, the likelihood score $l(c)$ for an alignment column $c$ ($c = 1 ... m$) can be calculated as follows, given the branch length $b_{vr}$ between nodes $i$ and $j$:

$$l(c) = \sum_{R=A}^{T}\left(\pi_R \vec{L}_R^{(i)}(c) \sum_{S=A}^{T} P_{RS}(b_{vr})\vec{L}_S^{(j)}(c)\right) \tag{25.2}$$

The overall score then is computed by summing over the per-column log-likelihood scores as indicated in Equation 25.3.

$$\mathrm{L}nL = \sum_{c=1}^{m} \log(l(c)) \tag{25.3}$$

An important property of the likelihood function is the assumption that sites evolve independently (*i.e.*, all entries $c$ of the probability vectors $\vec{L}$ can be computed independently). This property represents the main source of fine-grain parallelism in the PLF (see, *e.g.*, [41, 67, 68, 65, 66, 45]).

To compute the maximum likelihood value on a fixed tree topology, all individual branch lengths as well as the substitution rates in the $Q$ matrix and the $\alpha$ shape parameter of the $\Gamma$ distribution also must be optimized via an ML estimate. For the $Q$ matrix and the $\alpha$ shape parameter, the most common approach in state-of-the-art ML implementations consists of using Brent's algorithm [5]. A key computational issue is that to evaluate changes in $Q$ or $\alpha$, the entire tree needs to be retraversed (*i.e.*, a *full* tree traversal needs to be conducted from the leaves toward the virtual root to correctly recompute the likelihood under the modified model parameters). For the optimization of branch lengths, the Newton-Raphson method commonly is used. To optimize the branches of a tree, the branches are visited repeatedly and optimized one by one until the achieved likelihood improvement (or branch length change) is smaller than some predefined $\epsilon$. Because the branch length is optimized with respect to the likelihood score, the Newton-Raphson method only operates on a single pair of probability vectors $\vec{L}^{(i)}$, $\vec{L}^{(j)}$ that are located at either end of the branch that is being optimized. The Newton-Raphson method requires the computation of the first and second derivative of the likelihood function. Because we intend to maximize the likelihood function, we need to determine the root of the first derivative of the likelihood function. Evidently, when a branch of the tree is updated, this means that several probability vectors $\vec{L}$ in the tree are affected by this change and hence need to be recomputed to maintain a state that is consistent with the new branch length configuration.

An important implementation issue is the assignment of memory space for the probability vectors to inner nodes of the tree. Two alternative approaches exist; a separate vector can be assigned to each of the three outgoing branches of an inner node (PHYML uses this approach), or only one vector can be assigned to each inner node (GARLI, RAxML, and MrBayes, among others, deploy this technique). In the latter case, which is significantly more memory-efficient, the probability vectors always maintain a rooted view of the tree (*i.e.*, they are oriented toward the current virtual root of the tree). In the case that the virtual root then is relocated to a different branch (for instance, to optimize the respective branch length), a certain number of vectors, for which the orientation to the virtual root has changed by rerooting the tree, need to be recomputed. If the tree is traversed in an intelligent way (*e.g.*, for branch length optimization or subtree rearrangements (see Section 25.5)), then the number of probability vectors that will need to be recomputed after relocations of the virtual root can be minimized. An example for this type of data organization is provided in Figure 25.1. RAxML also uses this type of rooted probability vector organization to handle large-scale alignments because current phylogenomic datasets can require more than 100 Gb of main memory under the $\Gamma$ model of rate heterogeneity, even when using this efficient organization of the inner (ancestral) vectors.

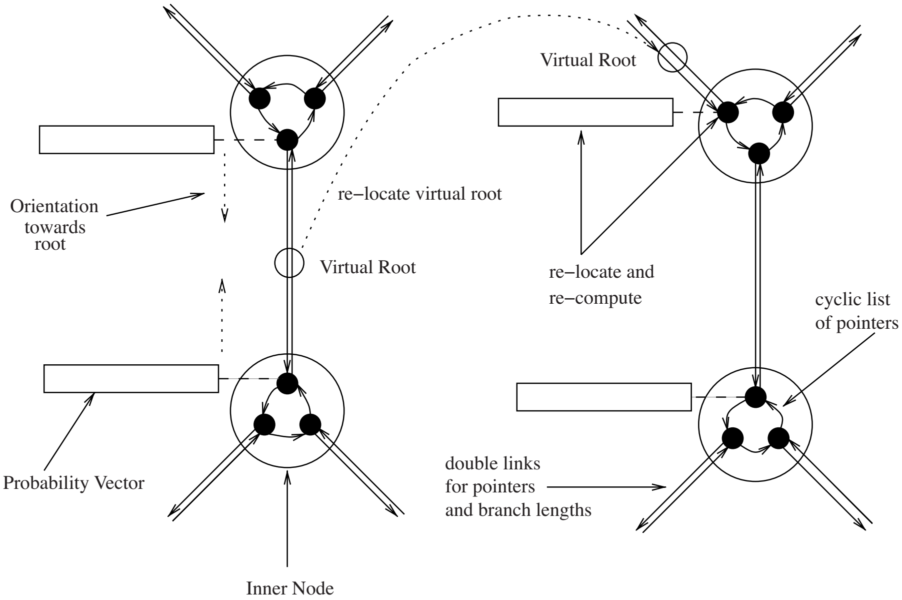


**Figure 25.1** Rooted organization of the probability vectors at inner nodes. This figure also shows the linked cyclic list of pointers at inner nodes and the double links between nodes.

## 25.3 ACCELERATING THE PLF BY ALGORITHMIC MEANS

### 25.3.1 Reuse of Values Across Probability Vectors

Some efforts have been reported [64, 63, 44] to accelerate the PLF by algorithmic means (*i.e.*, by detection and reuse of previously computed vector elements $p$ at a position $q$, where $q > p$, of a vector $\vec{L}$). If we look at Equation 25.1 again and consider computing all entries $c$ at once, then we may denote the left and right part as: $\vec{L}'^{(i)} = P(b_i) \times \vec{L}^{(i)}$ and $\vec{L}'^{(j)} = P(b_j) \times \vec{L}^{(i)}$ (*i.e.*, a dense matrix–matrix multiplication of the transition probability matrix with the probability vector). The vector $\vec{L}^{(k)}$ then is obtained by the element-wise multiplication of the entries in vectors $\vec{L}'^{(i)}$ and $\vec{L}'^{(j)}$. Clearly, the computations are dominated by the dense matrix–matrix multiplications. If we now consider the entries of vector $\vec{L}'^{(i)}$, for instance, then we can observe that two vector columns $p$ and $q$ of the respective subtree that are rooted at node $i$ will have the same entries if the nucleotide data at the leaves for the two columns $p$ and $q$ is identical. Thus, instead of recomputing the entries at $q$, we may just copy them from position $p$ in the same vector. The underlying concept is outlined in Figure 25.2.

Although this represents an interesting problem, also with respect to data structures that will help to keep track of identical column patterns in subtrees, several hardware-related and practical considerations actually have driven us to abandon

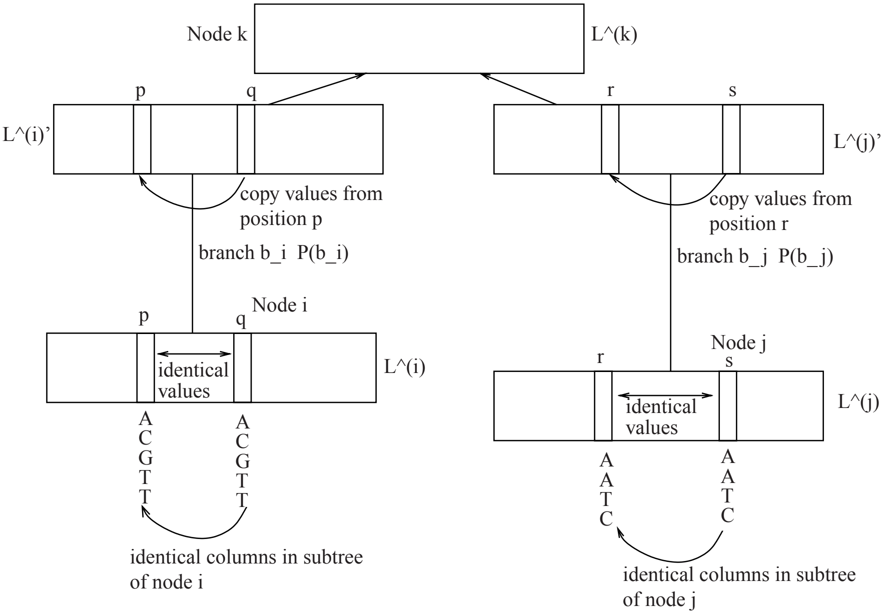


**Figure 25.2** Reusing values across probability vectors based on identical alignment patterns in respective subtrees.

completely these ideas in the current RAxML release. There are several reasons for not using this technique; first, these methods induce a form of irregular data accesses that may significantly perturb cache-efficiency compared with the straightforward linear computation of the probability vector entries. Thus, although we are saving floating point operations, we may encounter a significantly larger amount of cache misses. Second, an implementation of the PLF will require irregular accesses across a probability vector in the horizontal direction, which will yield the parallelization of the PLF at a fine-grained level (see [65, 41, 66]). This process is very difficult because the computation of columns $p$ and $q$ may not be independent from each other any more. Third, for these methods to be applicable, one must assume that all columns evolve under the same evolutionary model (*i.e.*, the CAT model of rate heterogeneity [57], a mixture model [31], or a partitioned model where ML parameters are estimated independently for different parts of the alignment are not used). In the case that vector entries $p$ and $q$ at a node $i$ have the same column pattern in the subalignment induced by the subtree rooted at $i$ but are assigned different evolutionary models because they have been assigned to different genes or rate categories, the values at positions $p$ and $q$ will *not* be identical. The use of partitioned and mixed models as well as of the CAT approximation represent the rule, rather then the exception in real-world phylogenetic analyses. Therefore, these algorithmic methods will probably not be of relevance for future search algorithms and PLF implementations. A potentially more promising algorithmic technique to accelerate the PLF is presented in the following section.

### 25.3.2 Gappy Alignments and Pointer Meshes

A current trend in phylogenetics goes toward phylogenomic alignments (*i.e.*, alignments that consist of a concatenation of many genes (100–1000 genes), with relatively few taxa (50–200 taxa)). An important property that characterizes these alignments is their gappiness (*i.e.*, a large part of the alignment, typically 50–95%, consists of gaps, or more precisely undetermined characters). Given two genes, $G_0$ and $G_1$ that contain a total of $n$ taxa, we usually will not have a gene sequence for both genes available for every taxon (*i.e.*, for some taxa, we only will have sequences for $G_0$ and for others only for $G_1$). The missing gene sequence in a taxon of a multigene datasets then is filled with undetermined characters that are mathematically exactly identical to gaps in the standard implementation of the ML function that is used in programs such as GARLI, MrBayes, IQPNNI, RAxML, and so on. Figure 25.3 provides an example for such a gappy multigene dataset with some missing sequence data in each gene.

Given the way gaps are modeled under ML, we can observe that adding a taxon that consists entirely of gaps to a tree at an arbitrary branch will not change its likelihood. If we conduct a partitioned analysis of the multigene dataset outlined in Figure 25.3, and we also apply a per-partition (per-gene) estimate of branch lengths, then we observe the following: For a given tree $t$ that comprises all five taxa, we may compute the overall likelihood as $\mathrm{Ln}L = \mathrm{Ln}L(t|G_0) + \mathrm{Ln}L(t|G_1)$, where $\mathrm{Ln}L(t|G_i)$ is the likelihood of the tree $t$ for gene $G_i$ *restricted* to the taxa for which we actually have sequence data available in gene $i$. In Figure 25.3, this means that we need to add the likelihoods of two three-taxon trees instead of two five-taxon trees for genes $G_0$ and $G_1$ under the standard implementation. This allows us to save a significant amount of floating point operations and also a significant amount of memory space for ancestral probability vectors. This memory footprint reduction is proportional to the gappiness of the respective alignment. For details and respective performance data, please refer to [65].

The key challenge consists of designing rapid methods to extract the subtrees induced by genes from the comprehensive tree $t$ and maintaining the data structures outlined in Figure 25.1 in a consistent state. This is both an algorithmic as well as a software engineering challenge because of the high complexity of this approach. We currently can read in and score trees (*i.e.*, optimize all ML model parameters

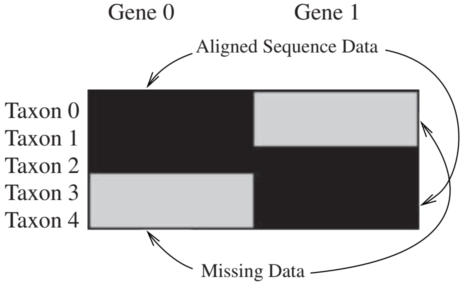


**Figure 25.3** A gappy multigene alignment.

except the tree topology) in RAxML by using appropriate static pointer meshes for representing and navigating through the per-gene subtrees (see [65] for details).

However, conducting actual searches with subtree pruning regrafting (SPR) moves (see Sections 25.5 and 25.5.1) means that we need to update dynamically those complex pointer meshes and require an efficient mechanism to derive quickly whether a specific SPR move induces a change on each individual per-gene subtree. We believe that this can be accomplished via a provably correct rule set that will allow to update dynamically the pointer mesh. Although the currently ongoing implementation is rather complicated and error-prone, we believe that this will represent an important mechanism to further accelerate phylogenetic inferences. This assumes of course, that phylogenomic datasets will remain gappy, which also depends on future developments in wet lab sequencing techniques, but it seems likely that the community will be facing these gappy alignments for at least another 5 years.

## 25.4 ALIGNMENT SHAPES

An important consideration for the design of search algorithms is to take the actual alignment shape into account (*i.e.*, whether one intends to analyze a datasets with many taxa and few genes or few genes and many taxa).

The two aforementioned large-scale phylogenetic studies [9, 54] point toward a fundamental problem that will need to be tackled in the future; the phylogenomic study [9] contains less than 100 taxa but 150 genes; an ongoing follow-up study [24] even comprises approximately 1000 genes. The study by Smith and Donoghue [54] is based on two datasets with less than ten genes but more than 4000 and 13,000 taxa, respectively. The datasets used in those two representative studies have significantly distinct shapes that impact algorithm design, scalability, as well as appropriate parallelization strategies. Here, I introduce the term "well-shaped" alignments for few-taxa/many-gene input datasets and "badly shaped" for many-taxa/few-gene datasets (see Figure 25.4). Evidently, badly shaped datasets are harder to analyze algorithmically and also are more difficult to parallelize than well-shaped alignments. In well-shaped datasets, we have, despite their gappiness (see Section 25.3.2), a large amount of data available to infer the evolutionary relationships among relatively few taxa (*i.e.*, the signal in the data is strong). This means that multiple searches for the best-scoring ML tree on distinct starting trees will yield trees that are related closely to each other in terms of their topological distance (see Section 25.6).

In contrast, badly shaped alignments exhibit a plethora of likelihood peaks that can not be distinguished from each other via statistical significance tests (*i.e.*, exhibit a "rough" likelihood surface with many likelihood peaks) and have large topological distances between each other (see also Section 25.7.1). This phenomenon also has been observed for studies on simulated datasets [36, 3], which clearly show that reconstruction accuracy increases with an increasing number of sites in the alignment. Thus, although it may be sufficient to infer *the* best-known ML tree for a well-shaped alignment, for a badly shaped alignment, one preferably should sample as many trees as possible from the "rough" likelihood surface and then summarize the information contained in those trees appropriately.

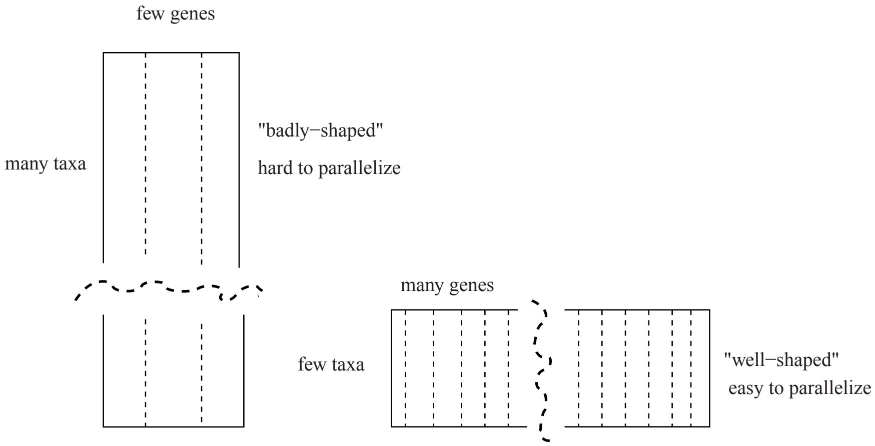


**Figure 25.4** Badly shaped and well-shaped input alignments.

As an example we will consider 20 ML trees produced by respective RAxML searches on two datasets, one with 125 taxa and 34 genes (well-shaped) and one single-gene dataset with 7764 taxa (badly shaped). The average relative topological distance between all pairs of 20 trees for the 125 taxon dataset was 0.5%, whereas the average topological distance between all pairs of 20 trees for the single-gene dataset was 33.80%.

The above example shows that we most likely will need to design specialized search algorithms that can handle more appropriately well-shaped and badly shaped alignments.

## 25.5 GENERAL SEARCH HEURISTICS

Initially, I will outline some basic search strategies that are used. Most modern search algorithms, such as GARLI, RAxML, IQPNNI, or PHYML, start their search for a best-known likelihood tree (the maximum likelihood tree is unknown because the problem is NP-hard) on a comprehensive starting tree (*i.e.*, a tree that contains all taxa). Such comprehensive starting trees can be obtained via some of the simpler and thereby less compute-intensive methods, for instance, via [14] MP or neighbor joining [50] (NJ). Sometimes random trees also are used as starting trees; however, the likelihood of a random starting tree is typically significantly lower than that of a reasonable starting tree obtained via MP or NJ. Thus, the use of reasonable starting trees can help to save a considerable amount of time in the inference process as outlined in Figure 25.5 for two tree inferences with RAxML on a 101 taxon single-gene DNA dataset under the GTR+Γ model. The MP starting tree has a significantly better initial likelihood than the random starting tree and requires more than 30% less time for the search to converge.

An advantage of MP starting trees over NJ starting trees is that for obtaining MP starting trees, a randomized stepwise addition order algorithm can be used to

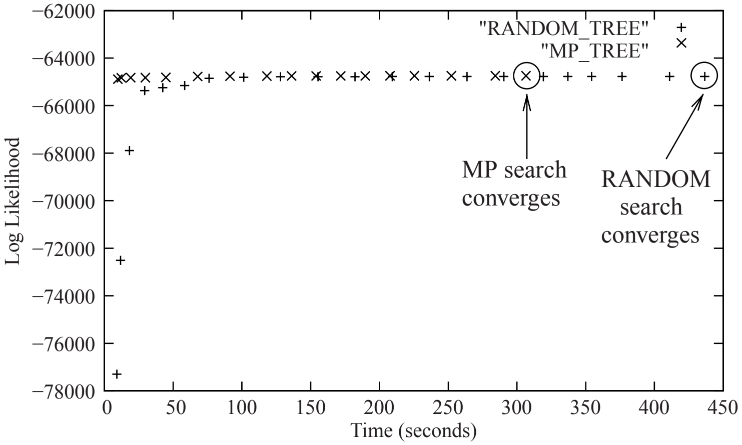


**Figure 25.5** Log likelihood over time plot for an ML analysis of a 101 taxon DNA dataset with RAxML using a maximum parsimony and a random starting tree.

generate a set of topologically distinct starting trees. Thus, one can conduct several searches to find the best-scoring/best-known ML tree by using distinct MP starting trees. Except for well-shaped alignments, searches on distinct starting trees typically will yield distinct ML trees. The advantage of this approach is that the enormous tree space can be explored more thoroughly as shown in Figure 25.6.

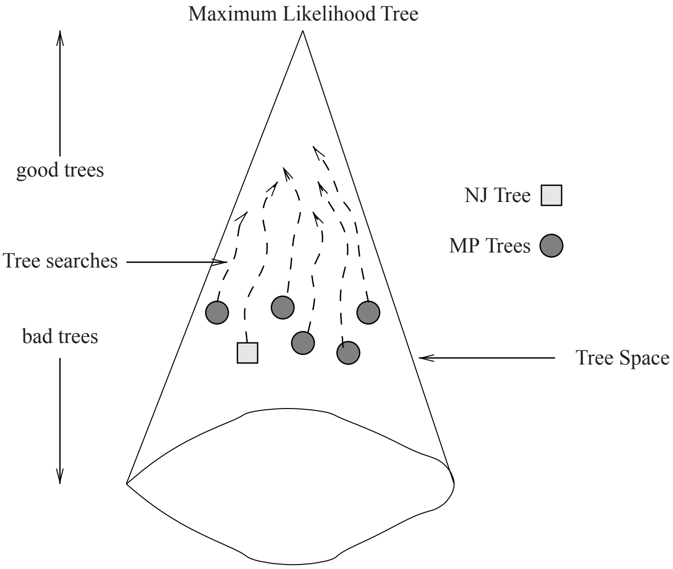


**Figure 25.6** Exploration of tree space under maximum likelihood using a neighbor joining starting tree and a set of randomized stepwise addition MP starting trees.

Because the stepwise addition algorithm is an important algorithm that also can be used for building comprehensive starting trees under ML as implemented, for example, in fastDNAml [39], I briefly will outline it. Given a set of $n$ taxa, one initially selects three taxa at random and builds the only possible unrooted binary tree of size three. Then, one of the remaining $n - 3$ taxa from the set is selected again at random and inserted into the tree of size three as follows: the new taxon is inserted (and removed again) into every branch of the current tree of size three, and the respective MP or ML score of the thereby obtained tree of size four is computed. The best of the three resulting trees of size four then is kept, and the fifth taxon once again is selected at random from the remaining $n - 4$ taxa. This fifth taxon then is inserted into all $2k - 3$ branches of the current tree (*i.e.*, into all $2 \times 4 - 3 = 5$ branches of the tree of size four). This procedure is continued until all taxa have been added to the tree. Thus, the stepwise addition algorithm can produce distinct starting trees for distinct sequence addition orders depending on the strength of the signal in the data. In the specific RAxML implementation, we also execute some further MP-based topological optimizations on the comprehensive tree. Extensive computational experiments have shown that this yields slightly better run times than just using the MP stepwise addition tree and then immediately applying ML-based topology optimization.

Once the comprehensive starting tree has been computed, one can start optimizing the ML score of the comprehensive tree by applying topological alteration mechanisms. This means that the topology of the currently best-scoring tree is altered, and then the likelihood of the new topology is evaluated. If a simple hill-climbing approach is used, then one may keep the newly generated topology if it has a better score and continue applying topological changes until no better tree can be found.

The three basic alteration mechanisms that are used are: nearest neighbor interchanges (NNI), subtree pruning and re-grafting (SPR), and tree bisection reconnection (TBR). A detailed description of NNI and TBR operators as well as a more detailed description of the stepwise addition algorithm can be found on pages 42–47 of [55] (available at http://wwwkramer.in.tum.de/exelixis/pubs/PHD.pdf). Because SPR moves are the most commonly used technique in present state-of-the art algorithms, I will outline this technique in more detail.

SPR moves are mostly used in ML search algorithms because they represent a good trade-off between the radicality/power of the topological change and the computational cost of scoring the altered tree topology (*i.e.*, the number of floating-point operations required to score the tree generated by a SPR move).

Given a tree, one can select a subtree that will be rearranged within the currently best tree via an application of SPR moves. Initially, this subtree will be pruned from the tree, and the branches of the thereby obtained smaller tree will need to be reoptimized under ML. To save some time, one also simply may reoptimize the branch at which the subtree was pruned. Then, one can start reinserting the pruned subtree into neighboring branches around the branch from which it was pruned originally. Those reinsertions either can be conducted up to branches that have a prespecified distance of nodes away from the original pruning position by using a so-called rearrangement setting or rearrangement radius. Alternatively, a pruned subtree can be inserted into

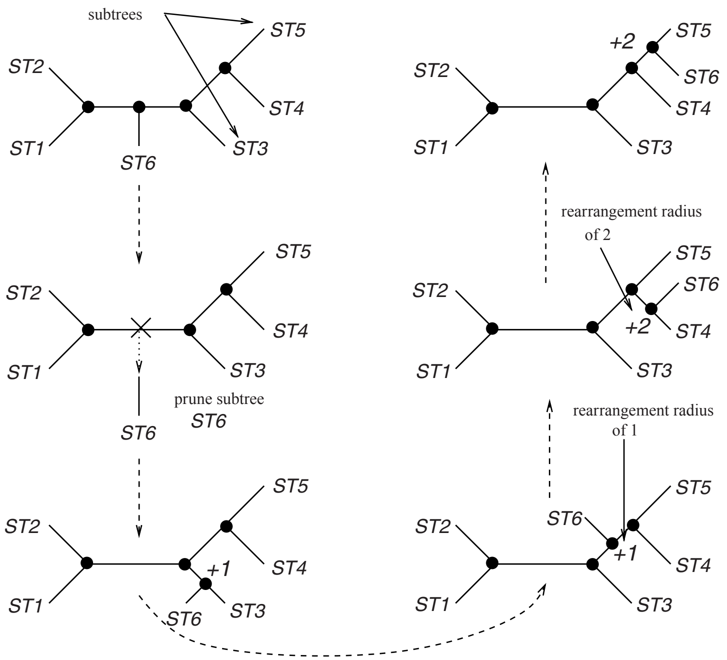


**Figure 25.7** Example for SPR moves applied to a subtree ST6 in a phylogenetic tree.

all branches of the tree it was pruned from. Evidently, the value of the rearrangement parameter has an impact on execution times as well as on the likelihood scores of the final trees (*i.e.*, there is a trade-off between speed and accuracy with respect to the rearrangement radius).

An example for the application of the SPR technique to a subtree ST6 is provided in Figure 25.7. SPR moves can be applied systematically to trees as follows: Given a current best-scoring tree, one can generate a list of all possible subtree roots in that tree. Thereafter, one can try to apply SPR moves to all subtrees within the current best tree, and once all of them have been executed, simply conduct the SPR move again that returned the largest likelihood improvement and thereby obtain a better tree. Alternatively, if a SPR move applied to a specific subtree already yields an improved likelihood score, then one immediately can keep the tree that was generated by the SPR move on the specific subtree and then execute the remaining SPR moves on the already modified tree. The latter technique is used in one form or the other in most modern search algorithms.

The systematic application of SPR moves to all possible subtrees in a given tree is called an SPR or rearrangement cycle. A computational challenge inherent to this approach is that, one would need to reoptimize all branch lengths in that tree to obtain the maximum likelihood score of the rearranged tree. Therefore, programs such as GARLI, RAxML, or PHYML use lazy SPR techniques, which are outlined in the following section.

### 25.5.1 Lazy Evaluation Strategies

As already mentioned, SPR moves are computationally significantly more expensive than, for instance, NNI moves, but they also yield trees with significantly better likelihood scores on real-world datasets (see [62, 58]). Therefore, additional measures need to be taken to accelerate SPR moves further. By now, a well-established technique consists of applying so-called lazy SPR moves. Lazy SPR moves are based on the rationale that branch lengths in a tree constructed via an SPR move only will change to a larger degree in an area that is close to the insertion point of the subtree that is being rearranged. Thus, it may suffice to readapt a few branch lengths that are in the vicinity of the insertion branch. In RAxML, this is implemented in a straightforward way by reoptimizing the three branches that are adjacent to the insertion position of the subtree that is being rearranged (see Figure 25.8). In addition, to save even more floating point operations, this lazy insertion comes in two flavors: fast lazy insertion in which simply good guesses for the three branch lengths are used and slow lazy insertions in which the three branch lengths are optimized via the standard Newton-Raphson procedure. The fast lazy optimization is used during the initial SPR cycles of RAxML (*i.e.*, to "get the big picture right"), whereas the slow and more thorough optimization is used for the latter SPR cycles to "get the details right".

GARLI uses a more sophisticated technique by reoptimizing branches located in an area around the insertion point. This area is determined dynamically (*i.e.*, the algorithm moves away from the insertion position and optimizes branch lengths until the branch length change is smaller than a certain threshold value).

Although these lazy evaluation procedures only yield *approximate* instead of *maximum* likelihood scores for the candidate trees that are constructed via the SPR technique, those scores then can be used to identify a set of "good" candidate topologies. Trees with approximate likelihood scores can be sorted by their scores, and thus, only a small fraction of promising trees needs to be assessed more thoroughly via exhaustive branch length optimization. For details, please refer to [62, 58] and [78] for the GARLI algorithm. Similar techniques also have been proposed for lazy SPRs

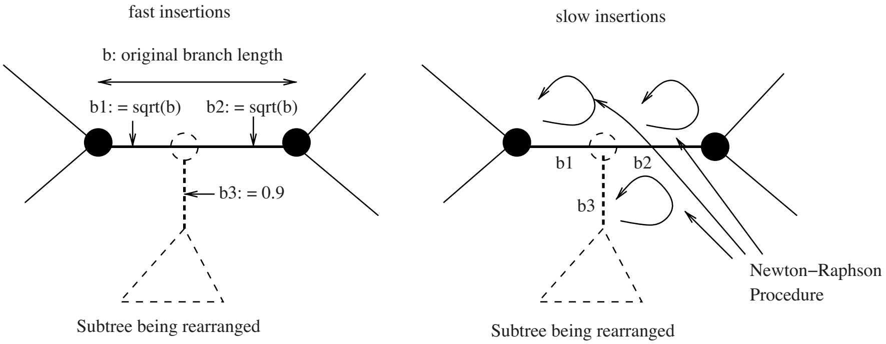


**Figure 25.8** Example of lazy local fast and slow branch length reoptimization in RAxML.

in PHYML [27] and there also have been some experiments using maximum parsimony, which is computationally significantly cheaper than ML, to prescore SPR-generated trees [61].

### 25.5.2 Further Heuristics

In a continuous strive to keep up with data accumulation, further heuristics recently have been integrated into RAxML [59] that are, however, not RAxML-specific (*i.e.*, they also could be used in other phylogenetic inference tools). The so-called likelihood cutoff heuristics allow for a reduction in the number of lazy SPR moves that are executed by omitting the computation of SPR moves that do not seem promising. The implementation of this technique yielded additional speed-ups of approximately a factor of two while returning equally good likelihood scores. The underlying idea consists of omitting subtree insertions into parts of the tree that do not seem to improve the likelihood score. This technique is outlined in Figure 25.9. Given a large rearrangement radius, of, for example, ten nodes, one only initially can consider the approximate likelihood scores of subtree insertions at a distance of one node from the original pruning position. The comparison of the score for those four alternative placements of the candidate subtree may provide an indication in which of the four directions (four branches) the likelihood may improve. Thus, for example, one can skip a further descent into a subtree that yields the worst of the four approximate

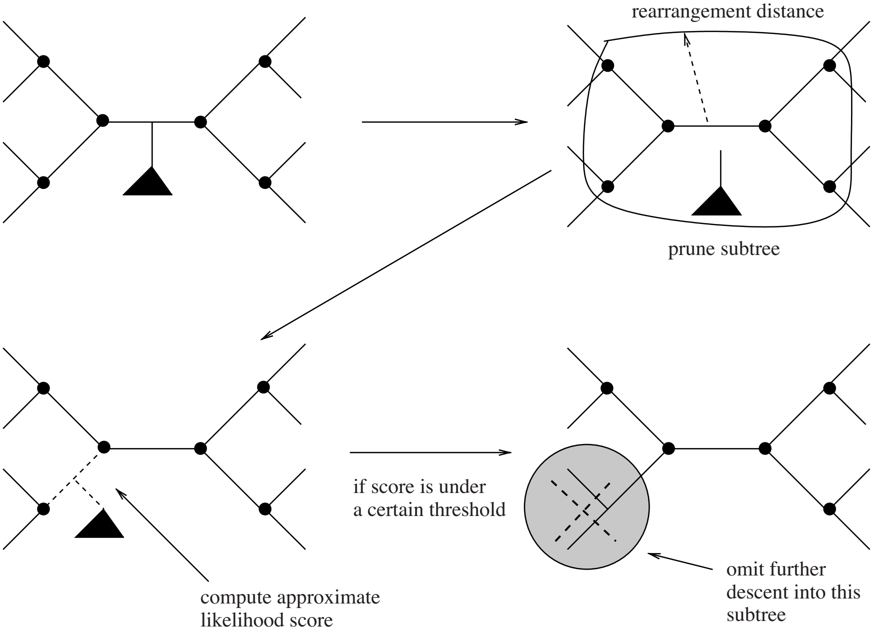


**Figure 25.9** Avoiding SPR descent into nonpromising parts of the tree via the likelihood cutoff technique.

likelihood scores and thereby avoid the computation of numerous lazy subtree rearrangements within that subtree. The decision, whether to descend further into a specific subtree is based on a dynamic threshold value. This value has been tuned on many real-world datasets and works well in practice. For further details, please refer to [59].

### 25.5.3 Rapid Bootstrapping

Another important issue with respect to program performance and inference times for real-world phylogenetic analyses are bootstrap analyses. The phylogenetic bootstrap procedure in phylogenetics was proposed by Joe Felsenstein in [13], and the general Bootstrap procedure was proposed by B. Efron in 1979 [11].

The phylogenetic bootstrap procedure works as follows: given an input alignment with $m$ columns/sites, we randomly draw $m$ columns from the original alignment to assemble a bootstrapped replicate alignment with a slightly different column composition. Typically, one will generate between $r = 100$ to $r = 1000$ bootstrap replicates from the original alignment. On each of the $r$ replicates, one usually applies the same ML search algorithm as applied to the original alignment and thereby will obtain a set of $r$ bootstrapped trees. Those $r$ trees then are used either to build a majority rule consensus tree of some flavor (see [28] for some alternative ways to build consensus trees) that roughly corresponds to computing an average. Alternatively, one can use the $r$ bootstrapped trees to compute support values for the branches (bipartitions see Section 25.6) of the best-known ML tree on the original alignment (*i.e.*, simply count how often a bipartition of the ML tree also is contained in the bootstrapped trees).

Thus, the major computational burden does not consist of "just" computing a ML tree but of conducting 100–1000 such searches on the respective bootstrap replicates. Typically, biologists need to make available phylogenetic trees with some form of support values (*e.g.*, bootstrap values or Bayesian posterior probabilities), to publish their evolutionary analyses. To solve this major computational bottleneck, we have developed a "quick and dirty" version of the RAxML search algorithm that is targeted at rapid computation of bootstrap replicates. This essentially can be regarded as algorithmic tuning of the underlying algorithms, by reusing trees or model parameters from previously generated replicates and reducing the number of SPR cycles per replicate. The accuracy of those approximate algorithms then was assessed thoroughly via computational experiments on numerous real-world benchmark datasets. For details on the algorithmic design and the respective performance assessment, please refer to [60].

Because the inference of support values generally is recognized as a computational bottleneck, alternative approaches to solving this problem have been suggested (*e.g.*, the approximate likelihood ratio test [aLRT [2]] that proposes a statistical framework for rapid computation of support values). The resampling of the estimated log likelihood (RELL) method also has been proposed as a means for accelerating the computation of support values [74], but our own (unpublished) experiments have shown that it is very difficult to deploy RELL for computing support values on large-scale datasets.

## 25.6 COMPUTING THE ROBINSON FOULDS DISTANCE

The efficient computation of the Robinson Foulds (RF) metric [47] is not only important for comparing phylogenetic trees with each other but also for integrating the ML search convergence criterion described in the following Section 25.7 and the bootstrap convergence criteria outlined in [42]. The main computational challenge is the design of efficient methods to extract, maintain, and operate on lists that contain all nontrivial bipartitions (splits) induced by a collection of trees. Apart from computing the RF distances, such lists of bipartitions also are required for computing consensus trees [28] or implementing convergence assessment mechanisms for Bayesian inference programs [38]. Although the theoretically optimal algorithm is well described [7], important technical details often are not considered and rarely are assessed experimentally, such as, for example, the choice of the hash function.

A nontrivial bipartition is a cut of a tree at an inner branch (*i.e.*, a branch that does not lead to a tip), into two disjoint sets of taxon labels, where each set contains $2 \leq i \leq n - 2$ taxon labels, and $n$ is the number of taxa in the tree. Because the sets induced by trivial bipartitions (*i.e.*, cuts/splits at branches that lead to a leave in a tree and contain 1 and $n - 1$ taxon labels, respectively), are contained in *all* possible trees, they do not carry any useful information about the actual tree topology and are hence discarded. Because an unrooted binary tree with $n$ leaves contains $2n - 3$ branches and $n$ branches thereof lead to leaves, a tree of size $n$ therefore induces $n - 3$ nontrivial bipartitions.

A bipartition can be represented by two presence/absence bit vectors $b_L, b_R$ of length $n$, where every bit denotes the presence/absence of a taxon in the subtree to the left ($b_L$) and to the right ($b_R$) of the branch that is being cut. Clearly, $b_L$ is the bit-wise complement of $b_R$. Because of this property, it suffices either to store $b_L$ or $b_R$. To ensure consistency of this choice between $b_R$ and $b_L$ and to avoid computational overhead for checking whether two bit vectors are bit-wise complements of each other, one may choose always to store the bit vector that contains (or does not contain) a specific taxon (*e.g.*, the first taxon in the input alignment). This is important to ensure consistency among bipartitions extracted from two distinct trees, $t_1$, $t_2$, because otherwise, a bipartition that is shared between the trees may be stored as $b_L$ for $t_1$ and as its complement $b_R$ for $t_2$.

Let us now consider how to extract efficiently bipartitions from an unrooted tree that already is stored in memory (*i.e.*, we do not discuss how to read in efficiently in trees in the standard NEWICK format [see http://evolution.genetics.washington.edu/phylip/newicktree.html] from file). The algorithm for efficient computation of the bipartitions at each inner branch is conceptually very similar to Felsenstein's pruning algorithm for computing the ML score on a tree and relies on a rooted view of the unrooted tree. Initially, we will assign bit vectors of length $n$ to all $2n - 2$ nodes of the tree and initialize the bipartition vectors at the tips (*i.e.*, just set the bit that corresponds to the respective taxon number). Thereafter, we place a virtual root into the branch that leads to the first taxon in the input alignment and recursively compute all bipartition vectors bottom-up toward the virtual root via a depth-first traversal. Keep in mind that all inner bipartition vectors will be oriented toward the virtual root of the tree. Every time we compute the bipartition vector at an inner node that is connected to another inner node, we can store directly the bipartition in a hash table. This means that we always are storing only those bipartitions that do not contain the selected taxon and thereby ensure consistency. The complexity of this operation is $O(n^2)$ because we need to compute $n - 3$ bipartition vectors and the computation of each bipartition vector is a for loop over $n$ bits. However, in practice 32, 64, or even 128 (if simple sharing extensions (SSE)-vectorized code is used) bit vector entries can be computed in one CPU cycle such that a more accurate approximation for the number of instructions is, for example, $n \times (n/32)$.

Given this efficient method for extracting bipartitions from trees, we now can consider the appropriate data structure for storing these bipartitions. The use of a hash table is straightforward and represents an efficient choice. However, the question is how to select a hash function for the hash key, which in our case, is simply the bipartition vector. The use of universal hash functions [6] as advocated in some more theoretical papers [70, 69, 1] may not represent the optimal choice. First, because the computation of a universal hash function given a bit vector of length $n$ is slow, and second universal hash functions only work well when hash keys equally are distributed randomly [6], which may not be the case for hash keys that are induced by a hierarchical data structure such as a tree.

We have assessed experimentally several highly tuned open-source hash functions that are nicely summarized at http://burtleburtle.net/bob/hash/doobs.html by adopting an algorithmic engineering approach [35]. In addition to this collection of hash functions, we also tested a phylogeny-specific hash key proposed by Pattengale *et al.* [43]. This method takes advantage of the tree structure and works as follows: as hash keys, we use 32- or 64-bit integer values instead of full-length bipartition bit vectors. Initially, each taxon will be assigned a random unsigned 64-bit integer number. Then, the hash numbers for the bipartitions also are computed bottom up toward the virtual root by performing a bit-wise exclusive or on the respective child vectors. This procedure can be integrated conveniently into the depth-first traversal that is used to compute the bipartition vectors. Extensive tests on large collections of trees have revealed that this method slightly outperforms all other tested hash functions in terms of speed and generates approximately the same amount of collisions that are resolved by chaining in the current RAxML implementation.

If we want to compute the RF distance, then we also need to keep track of which tree in the tree collection contained a bipartition that is stored in the hash table. For this, we deploy a presence/absence bit vector of length $x$, where $x$ is the number of trees. Hence, if we add an entry to the hash table and the respective slot is already occupied, then we initially need to compare the bipartition vector (or list of bipartition vectors) in that slot with the bipartition vector to be added. If it matches one of the stored bipartition vectors, then we simply set the respective bit for the tree number to one, otherwise we resolve by chaining.

The mechanisms described now can be deployed to store all nontrivial bipartitions of a set of $x$ trees that contain $n$ taxa. The Robinson Foulds distance is defined as the number of bipartitions that are unique to one of the two trees but not both. If $x = 2$, then we need to look at all valid entries of the hash table and increment a counter by

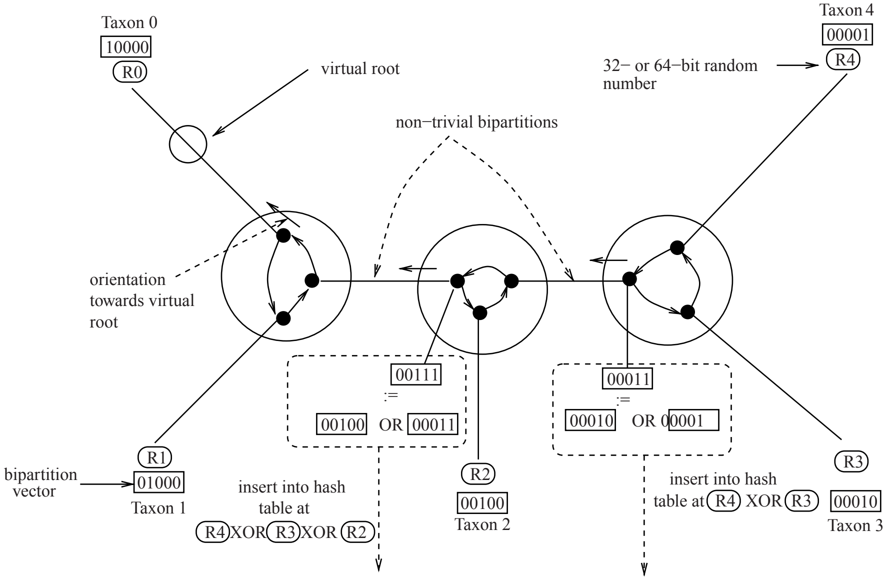


**Figure 25.10** Outline of the procedure to extract bipartitions efficiently and generate bipartition hash numbers on an unrooted binary tree.

one if the bit vector that contains the trees that generated the specific bipartition has one bit set to zero (*i.e.*, 01 or 10).

If we need to compute all pair-wise RF distances between a collection of $x$ trees, then the tree presence/absence bit vector needs to be analyzed as follows: If a bit $i$ is set to 1, then we need to find all other bits $j \neq i$ that are set to zero and increment the respective pair-wise RF distance between $i$ and $j$ by one. The complexity of this operation is $O(x^2)$ and can dominate the run times (75–80% of total execution time) for large collections of trees (*e.g.*, on 10,000 trees with 2000 taxa each). The specific RAxML implementation scales on trees of up to 37,000 taxa.

The procedure to extract bipartitions and generate hash numbers for bipartitions is outlined in Figure 25.10.

## 25.7 CONVERGENCE CRITERIA

Convergence criteria also can be seen as an algorithmic means to avoid unnecessary use of computational resources. The goal of these criteria is only to perform as many computations as necessary to achieve a certain accuracy level.

A question that comes up within the context of bootstrap analyses (see Section 25.5.3) is how many bootstrap replicates $r$ actually are required to obtain stable support values. Although Hedges discusses this issue for phylogenetics in a theoretical [23] context, we have proposed an empirical approach that takes into account the variability in the phylogenetic signal of different input dataset shapes. As may be expected, well-shaped alignments require less bootstrap replicates than badly shaped alignments to achieve stable support values. A detailed description of bootstrap convergence criteria is provided in [42]. Although these criteria allow for only conducting as many ML searches on bootstrap replicates as necessary and thereby help to economize on computational resources, one also may need to reconsider convergence properties of single, stand-alone ML searches as described in the following section.

### 25.7.1 Asymptotic Stopping

One common problem inherent to ML-based searches is that of asymptotic convergence of the log-likelihood score over time. Initially, the search algorithm will generate significant improvements in likelihood scores, but thereafter, it will reach a phase of slow asymptotic convergence that consumes the largest portion of execution time and only yields insignificant improvements in likelihood scores. This phenomenon is particularly extreme for badly shaped alignments. As already mentioned, one of the key characteristics of multiple ML searches (*e.g.*, on 100 randomized stepwise addition MP starting trees) on such badly shaped alignments is that they typically yield several ML trees that can not be distinguished statistically from each other. This means that they are not significantly different from each other based on standard statistical significance tests (see [18] for a summary of such tests). Hence, one most likely will need to infer rapidly a large collection of equally “good” ML trees that then will need to be summarized by appropriate consensus tree techniques.

Given the prolegomena, it is not clear whether we really require the small improvements in the likelihood score that are obtained during this asymptotic convergence phase. It may be preferable to compute rapidly a larger number of ML trees and then summarize them via consensus techniques. In a current large-scale study on real data, we found that on an alignment with six genes and 37,831 taxa the 25 ML trees we had computed showed an average pair-wise RF distance of approximately 20%, whereas the trees could not be distinguished statistically from each other by their ML scores.

An example for asymptotic convergence behavior is outlined in Figure 25.11 for a single-gene dataset with 1303 base-pairs (sites) and 34,584 taxa analyzed under GTR+Γ using 16 threads on an AMD Barcelona multicore system. The plot shows two plateaus before the final asymptotic convergence phase, which are a result of the algorithmic design of RAxML (the transition from SPR cycles with fast insertion to slow insertion, see Section 25.5.1), that is, those plateaus are predictable. The graph clearly shows that, if the tree search is stopped early during the final asymptotic convergence phase or convergence plateau, then around 70% of total execution time can be saved.

In the following paragraphs, I briefly will describe a novel convergence criterion for RAxML searches that has been tested empirically on 12 single-gene (*i.e.*, badly shaped) real-world datasets, comprising 1288–4114 taxa. The criterion only is applied to SPR cycles that use the slow lazy insertion method for subtrees (see Section 25.5.1). The convergence criterion works as follows: For two successive cycles

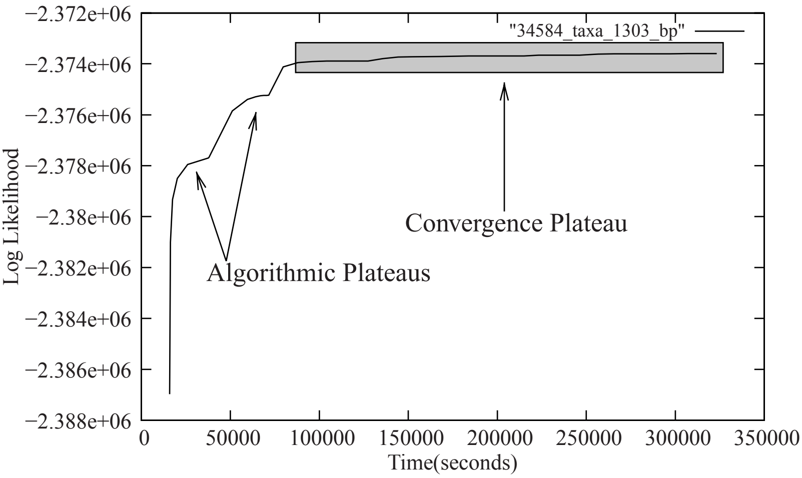


**Figure 25.11** Log-likelihood improvement over time for a single-gene DNA dataset with 34,584 taxa and 1303 base pairs. Phylogenetic search under the GTR+Γ model using the current development version of RAxML. The analysis was conducted with 16 threads on an AMD Barcelona multicore system with 16 cores and 128 Gb of main memory. Memory footprint of this analysis is 7.5 Gb.

of slow lazy SPR moves $c_i, c_{i+1}$, we keep track of the respective best-scoring trees $t_i, t_{i+1}$ produced by those cycles. We stop the ML search, if RF $(t_i, t_{i+1}) < 1\%$ (*i.e.*, if the topological distance between the trees produced by two successive SPR cycles is small). Evidently, there is a danger of stopping searches too early and obtaining trees with bad likelihood scores.

To assess the accuracy, or rather the loss of accuracy, caused by this convergence rule, we inferred 40 ML trees on the same set of MP starting trees for all 12 alignments with (denoted as STOP) and without (denoted as FULL) the stopping criterion. We conducted 960 tree searches using the Pthreads-based version of RAxML on a four-core Intel Core-2 Quad system running at 2.83 GHz under the GTR + CAT approximation of rate heterogeneity [57]. For all collections of trees, we then computed the likelihood scores under the standard GTR + Γ model of rate heterogeneity. In Table 25.1 we indicate the likelihood score for the best out of the 40 trees for searches using the stopping rule (column: LnL-STOP) and full searches (column: LnL-FULL). We also indicate the average log likelihood of all 40 trees for the stopped (column: Avg. LnL-STOP) and full searches (column: Avg. LnL-FULL). As can be derived from this table, the best and average likelihood scores obtained by the algorithm with the stopping rule are only slightly worse than those obtained by the full algorithm.

In Table 25.2, we indicate the execution times in hours for the STOP method (STOP Time(hrs)) and the FULL method (FULL Time (hrs)) as well as the respective speedup achieved by using the stopping criterion (Speed-up). Finally, we also indicate the RF distance between the respective best trees obtained by each method. The STOP method achieves an average speedup of 1.76. This means that in the time

**Table 25.1 Best and average likelihood scores returned by 40 distinct ML searches using the full search algorithm (FULL) and the search algorithm with stopping criterion (STOP)**

| # Taxa | LnL-STOP | LnL-FULL | Avg. LnL-STOP | Avg. LnL-FULL |
|---|---|---|---|---|
| 1288 | −395,860.48 | −395,849.25 | −396,020.61 | −396,016.14 |
| 1481 | −197,409.81 | −197,409.88 | −197,589.92 | −197,577.66 |
| 1604 | −167,336.65 | −167,312.87 | −167,381.09 | −167,372.03 |
| 1908 | −149,595.77 | −149,595.79 | −149,626.61 | −149,622.75 |
| 2000 | −364,871.78 | −364,856.96 | −364,925.20 | −364,894.23 |
| 2200 | −179,613.35 | −179,609.35 | −179,631.02 | −179,627.14 |
| 2308 | −449,803.17 | −449,803.32 | −449,910.36 | −449,898.68 |
| 2586 | −162,917.75 | −162,897.54 | −162,973.47 | −162,957.46 |
| 2843 | −143,187.96 | −143,180.69 | −143,227.51 | −143,218.72 |
| 2884 | −173,644.22 | −173,643.32 | −173,685.98 | −173,678.72 |
| 3564 | −389,749.24 | −389,738.73 | −389,894.42 | −389,848.05 |
| 4114 | −325,512.71 | −325,426.77 | −325,662.86 | −325,605.34 |

required to infer 40 trees using the FULL method, 70 trees could be inferred via the STOP method. Thus, given the preceding arguments, we believe that the same amount of computational resources can be used in a more efficient way by inferring more trees, albeit with slightly worse likelihood scores. This will help to explore better the likelihood surface. Finally, we also computed the RF distance between the respective best-scoring trees returned by FULL and STOP. Despite the apparently small and insignificant differences in likelihood scores, the topological distances between the trees are surprisingly high.

To investigate further this phenomenon, we computed the average pairwise RF-distances between all 40 trees obtained via the FULL method and all 40 ML trees obtained via the STOP method. In Figure 25.12, we plot those average distances over

**Table 25.2 Execution times, speed-ups, and RF distance between respective best trees for the full algorithm and the algorithm with stopping rule**

| # Taxa | STOP Time(hrs) | FULL Time (hrs) | Speed-up | RF Distance |
|---|---|---|---|---|
| 1288 | 12.32 | 21.41 | 1.74 | 2.3 |
| 1481 | 17.25 | 21.64 | 1.25 | 1.2 |
| 1604 | 11.86 | 18.84 | 1.59 | 16.6 |
| 1908 | 14.09 | 22.65 | 1.60 | 2.7 |
| 2000 | 20.92 | 43.30 | 2.07 | 23.4 |
| 2200 | 18.17 | 27.54 | 1.52 | 12.4 |
| 2308 | 20.29 | 35.25 | 1.74 | 0.6 |
| 2586 | 22.58 | 45.35 | 2.01 | 18.4 |
| 2843 | 28.48 | 51.06 | 1.79 | 4.3 |
| 2884 | 25.89 | 44.87 | 1.73 | 1.2 |
| 3564 | 56.22 | 107.63 | 1.91 | 2.9 |
| 4114 | 41.44 | 89.51 | 2.16 | 30.7 |

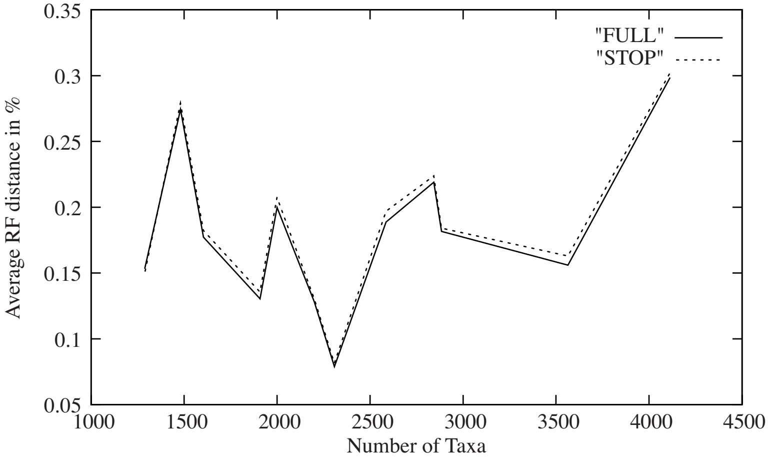


**Figure 25.12** Average pair-wise RF distances between the 40 distinct ML trees inferred by the full method and the method with ML convergence criterion.

the number of taxa. Although there seems to be a large dataset-dependent variation with respect to the RF distances, this plot demonstrates that the values obtained in Table 25.2 are representative of the distances that may occur between trees that do not have significantly different likelihood scores on badly shaped alignments. Overall, the proposed method seems to provide a useful means and a better way to allocate and use computational resources for exploration of the tree space of badly shaped alignments.

## 25.8 FUTURE DIRECTIONS

As has been mentioned frequently in this chapter, algorithmic techniques for phylogenetic inference under ML will have to evolve with or rather be adapted to alignment shapes.

With respect to well-shaped alignments and because of the strength of the phylogenetic signal contained in these, it may be worthwhile to assess whether equally accurate trees can be obtained, for example, by using simpler and less computational as well as memory-intensive methods such as NJ or MP. One also may consider to apply less radical/exhaustive topological moves under ML to such datasets. Moreover, one should not neglect issues that are associated with parallel computing and the multicore revolution. Large multicore nodes with 16–32 cores and even supercomputers such as the IBM BlueGene/L [40] may be required to handle the largest and most challenging phylogenomic analyses. Unfortunately, algorithm design issues and parallelization problems can not be separated entirely from each other. Thus, algorithmic design also should take into consideration future supercomputer architectures. Because of the constant increase in CPU count, scalability of current methods will be limited by Amdahl's law. Thus, it will be highly beneficial to design algorithms that can scale easily on several thousands of processors. A major algorithmic and software engineering challenge will consist of conducting tree searches under the adapted ML method described in Section 25.3.2. An implementation of searches under this method will contribute significantly to keeping pace with the data accumulation.

Overall, the design of phylogenetic search algorithms seems to have evolved more into a discipline of trade-off engineering (*i.e.*, we need to answer the question of how we can exploit best a limited amount of computational resources to achieve the most accurate results). Examples for such an engineering approach to phylogenetics are the bootstopping and ML search stopping criteria that are described in Section 25.7.

With respect to badly shaped alignments, a huge need also seems to exist for novel algorithmic approaches, aside from the fact that parallelization is also significantly more challenging. For example, within the framework of the National Science Foundation funded plant tree of life cyberinfrastructure project (see http://chac.iplantcollaborative.org/documents/iptol.pdf), we intend to analyze a badly shaped alignment of approximately 500,000 taxa comprising all species of the green plants. Although RAxML currently can handle datasets up to 50,000–60,000 taxa, new algorithmic approaches will be required to handle the lack of signal in these datasets and sufficiently sample the rough likelihood surface. One may think about a reassessment of likelihood ratchet techniques [73], simulated annealing algorithms [56, 51], or the zoom-in/zoom-out technique that was introduced in the PhyNav program [32], which allows for a significant reduction of the number of taxa in the tree.

Therefore, despite the fact that phylogenetics have come of age, the wet lab sequencing and multicore revolutions offer many fascinating challenges that wait to be addressed and solved.

## REFERENCES

1. N. Amenta, F. Clarke, and K. StJohn. A linear-time majority tree algorithm. *Lect Notes Comput Sci*, 2812:216–227, 2003.
2. M. Anisimova and O. Gascuel. Approximate likelihood-ratio test for branches: A fast, accurate, and powerful alternative. *Syst Biol*, 55(4):539–552, 2006.
3. O.R.P. Bininda-Emonds, S.G. Brady, M.J. Sanderson, and J. Kim. Scaling of accuracy in extremely large phylogenetic trees. *Pacific Symposium on Biocomputing*, 2000, pp. 547–558.
4. B. Boussau and M. Gouy. Efficient likelihood computations with nonreversible models of evolution. *Syst Biol*, 55(5):756–768, 2006.
5. R.P. Brent. *Algorithms for Minimization Without Derivatives*. Prentice Hall, Englewood Cliggs, NJ, 1973.
6. J.L. Carter and M.N. Wegman. Universal classes of hash functions (Extended Abstract). *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, ACM, New York, 1977, pp. 106–112.

7. W.H.E. Day. Optimal algorithms for comparing trees with labeled leaves. *J Classif*, 2(1):7–28, 1985.
8. A.J. Drummond and A. Rambaut. BEAST: Bayesian evolutionary analysis by sampling trees. *BMC Evol Biol*, 7(214):1471–2148, 2007.
9. C.W. Dunn, A. Hejnol, D.Q. Matus, K. Pang, W.E. Browne, S.A. Smith, E. Seaver, G.W. Rouse, M. Obst, G.D. Edgecombe, M.V. Sorensen, S.H.D. Haddock, A. Schmidt-Rhaesa, A. Okusu, R.M. Kristensen, W.C. Wheeler, M.Q. Martindale, and G. Giribet. Broad phylogenomic sampling improves resolution of the animal tree of life. *Nature*, 452(7188):745–749, 2008.
10. A.W.F. Edwards, L.L. Cavalli-Sforza, V.H. Heywood, and J. McNeill. Phenetic and phylogenetic classification. *Publ Systemat Assoc*, 6:67–76, 1963.
11. B. Efron. Bootstrap methods: another look at the kackknife. *Ann Stat*, 7:1–26, 1979.
12. J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *J Mol Evol*, 17:368–376, 1981.
13. J. Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39(4):783–791, 1985.
14. W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(3760): 279–284, 1967.
15. R. Fleissner, D. Metzler, and A.V. Haeseler. Simultaneous statistical multiple alignment and phylogeny reconstruction. *Syst Biol*, 54:548–561, 2005.
16. L.R. Foulds and R.L. Graham. The Steiner problem in phylogeny is NP-complete. *Adv Appl Math*, 3(43-49):299, 1982.
17. L. Ganzert, G. Jurgens, U. Munster, and D. Wagner. Methanogenic communities in permafrost-affected soils of the laptev sea coast, siberian arctic, characterized by 16s rrna gene fingerprints. *FEMS Microbiol Ecol*, 59(2):476–488, 2007.
18. N. Goldman, J.P. Anderson, and A.G. Rodrigo. Likelihood-based tests of topologies in phylogenetics. *Syst Biol*, 49(4):652–670, 2000.
19. N. Goldman and Z. Yang. A codon-based model of nucleotide substitution for protein-coding DNA sequences. *Mol Biol Evol*, 11(5):725–736, 1994.
20. M. Gottschling, A. Stamatakis, I. Nindl, E. Stockfleth, A. Alonso, L. Gissmann, and I.G. Bravo. Multiple evolutionary mechanisms drive papillomavirus diversification. *Mol Biol Evol*, 2007.
21. G.W. Grimm, S.S. Renner, A. Stamatakis, and V. Hemleben. A nuclear ribosomal DNA phylogeny of acer inferred with maximum likelihood, splits graphs, and motif analyses of 606 sequences. *Evol Bioinformatics Online*, 2:279–294, 2006.
22. S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst Biol*, 52(5):696–704, 2003.
23. S.B. Hedges. The number of replications needed for accurate estimation of the bootstrap P value in phylogenetic studies. *Mol Biol Evol*, 9(2):366–369, 1992.
24. A. Hejnol, M. Obst, A. Stamatakis, M. Ott, G.W. Rouse, G.D. Edgecombe, P. Martinez, J. Baguna, X. Bailly, U. Jondelius, M. Wiens, W.E.G. Müller, E. Seaver, W.C. Wheeler, M.Q. Martindale, G. Giribet, and C.W. Dunn. Rooting the bilaterian tree with scalable phylogenomic and supercomputing tools. 2009. To appear.
25. T.H. Ogden and M.S. Rosenberg. Multiple sequence alignment accuracy and phylogenetic inference. *Syst Biol*, 55:314–328, 2006.

26. M. Höhl and M.A. Ragan. Is multiple sequence alignment required for accurate inference of phylogeny? *Syst Biol*, 56(2):206–221, 2007.

27. W. Hordijk and O. Gascuel. Improving the efficiency of SPR moves in phylogenetic tree search methods based on maximum likelihood. *Bioinformatics*, 21(24):4338–4347, 2005.

28. L.S. Jermiin, G.J. Olsen, K.L. Mengerson, and S. Easteal. Majority-rule consensus of phylogenetic trees obtained by maximum-likelihood analysis. *Mol Biol Evol*, 14(12):1296, 1997.

29. G. Jobb, A.V. Haeseler, and K. Strimmer. TREEFINDER: A powerful graphical analysis environment for molecular phylogenetics. *BMC Evol Biol*, 4, 2004.

30. N. Lartillot, S. Blanquart, and T. Lepage. PhyloBayes. v2. 3, 2007.

31. N. Lartillot and H. Philippe. A Bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. *Mol Biol Evol*, 21(6):1095–1109, 2004.

32. S.V. Le, H.A. Schmidt, and A.V. Haeseler. PhyNav: A novel approach to reconstruct large phylogenies. *Proceedings of GfKl Conference*, 2004.

33. A. Loytynoja and N. Goldman. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, 320(5883):1632, 2008.

34. B.Q. Minh, L.S. Vinh, A.V. Haeseler, and H.A. Schmidt. pIQPNNI: Parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21(19):3794–3796, 2005.

35. B.M.E. Moret. Towards a discipline of experimental algorithmics. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges: Papers Related to the DIMACS Challenge on Dictionaries and Priority Queues (1995-1996) and the DIMACS Challenge on Near Neighbor Searches (1998-1999)*, 2002, p. 197.

36. B.M.E. Moret, U. Roshan, and T. Warnow. Sequence-length requirements for phylogenetic methods. *Lect Notes Comput Sci*, 2452:343–356, 2002.

37. D.A. Morrison. Increasing the efficiency of searches for the maximum likelihood tree in a phylogenetic analysis of up to 150 nucleotide sequences. *Syst Biol*, 56(6):988–1010, 2007.

38. J.A.A. Nylander, J.C. Wilgenbusch, D.L. Warren, and D.L. Swofford. AWTY(are we there yet?): A system for graphical exploration of MCMC convergence in Bayesian phylogenetics. *Bioinformatics*, 24(4):581, 2008.

39. G.J. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek. fastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Bioinformatics*, 10(1):41–48, 1994.

40. M. Ott, J. Zola, S. Aluru, A.D. Johnson, D. Janies, and A. Stamatakis. Large-scale phylogenetic analysis on current HPC architectures. *Sci Program*, 16(2-3):255–270, 2008.

41. M. Ott, J. Zola, S. Aluru, and A. Stamatakis. Large-scale maximum likelihood-based phylogenetic analysis on the IBM bluegene/L. *Proceedings of IEEE/ACM Supercomputing Conference 2007 (SC2007)*, 2007.

42. N.D. Pattengale, M. Alipour, O.R.P. Bininda-Emonds, B.M.E. Moret, and A. Stamatakis. How many bootstrap replicates are necessary? *Proceedings of RECOMB2009*, 2009. To appear.

43. N.D. Pattengale, E.J. Gottlieb, and B.M.E. Moret. Efficiently computing the Robinson-Foulds metric. *J Comput Biol*, 14(6):724–735, 2007. PMID: 17691890.

44. S.L.K. Pond and S.V. Muse. Column sorting: Rapid calculation of the phylogenetic likelihood function. *Syst Biol*, 53(5):685–692, 2004.

45. F. Pratas, P. Trancoso, A. Stamatakis, and L. Sousa. Fine-grain parallelism for the phylogenetic likelihood functions on multi-cores, Cell/BE, and GPUs. 2009. To appear.

46. B. Redelings and M. Suchard. Joint Bayesian estimation of alignment and phylogeny. *Syst Biol*, 54(3), 2005.

47. D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Math Biosci*, 53(1-2):131–147, 1981.

48. S. Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans Comput Biol Bioinform*, 3(1):92–94, 2006.

49. F. Ronquist and J.P. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003.

50. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4:406–425, 1987.

51. L.A. Salter and D.K. Pearl. Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. *Syst Biol*, 50(1):7–17, 2001.

52. N.J. Savill, D.C. Hoyle, and P.G. Higgs. RNA sequence evolution with secondary structure constraints: Comparison of substitution rate models using maximum-likelihood methods. *Genetics*, 157:399–411, 2001.

53. J. Shendure and H. Ji. Next-generation DNA sequencing. *Nat Biotechnol*, 26(10):1135–1145, 2008.

54. S.A. Smith and M.J. Donoghue. Rates of molecular evolution are linked to life history in flowering plants. *Science*, 322(5898):86–89, 2008.

55. A. Stamatakis. *Distributed and Parallel Algorithms and Systems for Inference of Huge Phylogenetic Trees Based on the Maximum Likelihood Method*. PhD thesis, Technische Universität München, Germany, October 2004.

56. A. Stamatakis. An efficient program for phylogenetic inference using simulated annealing. *Proceedings of IPDPS2005*, HICOMB Workshop, Denver, Colorado, April 2005.

57. A. Stamatakis. Phylogenetic models of rate heterogeneity: A high performance computing perspective. *Proceedings of IPDPS2006*, HICOMB Workshop, Rhodos, Greece, April 2006.

58. A. Stamatakis. RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

59. A. Stamatakis, F. Blagojevic, C.D. Antonopoulos, and D.S. Nikolopoulos. Exploring new search algorithms and hardware for phylogenetics: RAxML meets the IBM cell. *J VLSI Sig Proc Syst*, 48(3):271–286, 2007.

60. A. Stamatakis, P. Hoover, and J. Rougemont. A rapid bootstrap algorithm for the RAxML web servers. *Syst Biol*, 57(5):758–771, 2008.

61. A. Stamatakis, T. Ludwig, and H. Meier. A fast program for maximum likelihood-based inference of large phylogenetic trees. *Proceedings of 19th ACM Symposium on Applied Computing (SAC2004)*, 2004, pp. 197–201.

62. A. Stamatakis, T. Ludwig, and H. Meier. RAxML-III: A fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, 2005.

63. A. Stamatakis, T. Ludwig, H. Meier, and M.J. Wolf. Accelerating parallel maximum likelihood-based phylogenetic tree calculations using subtree equality vectors. *Proceedings of IEEE/ACM Supercomputing Conference 2002 (SC2002)*, 2002.

64. A. Stamatakis, T. Ludwig, H. Meier, and M.J. Wolf. AxML: A fast program for sequential and parallel phylogenetic tree calculations based on the maximum likelihood method. *Proceedings of 1st IEEE Computer Society Bioinformatics Conference (CSB2002)*, 2002, pp. 21–28.

65. A. Stamatakis and M. Ott. Efficient computation of the phylogenetic likelihood function on multi-gene alignments and multi-core architectures. *Phil Trans R Soc B Biol Sci*, 363:3977–3984, 2008.

66. A. Stamatakis and M. Ott. Exploiting fine-grained parallelism in the phylogenetic likelihood function with MPI, Pthreads, and OpenMP: A performance study. In M. Chetty, A. Ngom, and S. Ahmad, editors, *PRIB*, volume 5265 of *Lecture Notes in Computer Science*, Springer, New York, 2008, pp. 424–435.

67. A. Stamatakis and M. Ott. Load balance in the phylogenetic likelihood kernel. *Proceedings of ICPP 2009*, 2009. To appear.

68. A. Stamatakis, M. Ott, and T. Ludwig. RAxML-OMP: An efficient program for phylogenetic inference on SMPs. *PaCT*, 2005, pp. 288–302.

69. S.J. Sul, G. Brammer, and T.L. Williams. Efficiently computing arbitrarily-sized Robinson-Foulds distance matrices. *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics*, Springer, New York, 2008, pp. 123–134.

70. S.J.I.N. Sul and T.L. Williams. A randomized algorithm for comparing sets of phylogenetic trees. *Proceedings of the 5th Asia-Pacific Bioinformatics Conference: Hong Kong, 15-17 January 2007*, Imperial College Pr, 2007, p. 121.

71. D.L. Swofford. *PAUP*: Phylogenetic Analysis using Parsimony (* and other methods), version 4.0b10*. Sinauer Associates, 2002.

72. S. Tavaré. Some probabilistic and statistical problems in the analysis of DNA sequences. *In American Mathematical Society: Lectures on Mathematics in the Life Sciences*, 17:57–86, 1986.

73. R.A. Vos. Accelerated likelihood surface exploration: The likelihood ratchet. *Syst Biol*, 52(3):368–373, 2003.

74. P.J. Waddell, H. Kishino, and R. Ota. Very fast algorithms for evaluating the stability of ML and Bayesian phylogenetic trees from sequence data. *Genome Informatics Series*, 2002, pp. 82–92.

75. W. Wheeler, L. Aagesen, C.P. Arango, J. Faivovich, T. Grant, C. D'Haese, D. Janies, W.L. Smith, A. Varon, and G. Giribet. *Dynamic Homology and Phylogenetic Systematics: A Unified Approach using POY*. American Museum of National History, 2006.

76. Z. Yang. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites. *J Mol Evol*, 39:306–314, 1994.

77. Z. Yang. PAML 4: Phylogenetic analysis by maximum likelihood. *Mol Biol Evol*, 24(8):1586, 2007.

78. D. Zwickl. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion*. PhD thesis, University of Texas at Austin, April 2006.

# 26

# HEURISTIC METHODS FOR PHYLOGENETIC RECONSTRUCTION WITH MAXIMUM PARSIMONY

Adrien Goëffon, Jean-Michel Richer, and Jin-Kao Hao

*In this chapter, we explain how metaheuristics like local search, genetic, and memetic algorithms are used for phylogenetic reconstruction using maximum parsimony. We review some main concepts used to improve the search of a good solution that are inherited from the operational research and combinatorial optimization communities.*

## 26.1 INTRODUCTION

Maximum parsimony (MP) is a character-based approach that relies on the work of the german entomologist Willy Hennig (1913–1976). Although Hennig's work has generated significant controversy, the principles that underlie what was later called cladistics laid the basis for a convenient and powerful method for the analysis of molecular data with the use of computers. For more details about the early history of MP methods, see [9] (p. 136).

Cladistics, also referred to as phylogenetic systematics, can be viewed as a philosophy of classification that arranges organisms only by their order of branching in an evolutionary tree. The leaves of the tree are labeled with the operational taxonomic unit (OTU) of the problem also called taxa (singular: taxon). Ideally, the trees

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

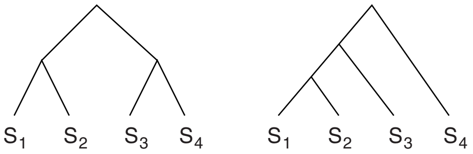


**Figure 26.1** Two tree topologies for a binary rooted tree of four sequences.

(or cladograms) that result from an MP analysis show the evolution of synapomorphies (derived character states inherited from the most common ancestor) between species. Many different cladograms can exist for a given set of taxa, but the MP criterion imposes to choose the ones with the fewest changes.

## 26.2 DEFINITIONS AND FORMAL BACKGROUND

### 26.2.1 Parsimony and Maximum Parsimony

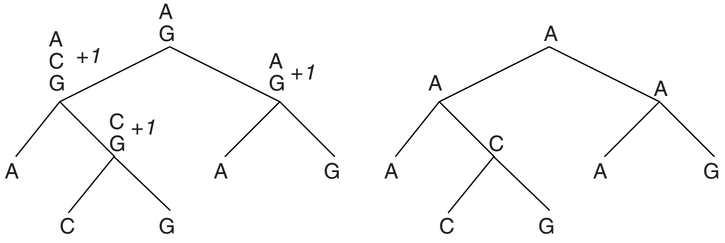
With parsimony methods, each position (or site) in the multiple alignment is considered separately. First note that there are different parsimony optimality criteria known as Fitch, Wagner, Camin-Sokal, Dollo, or weighted parsimony. Those criteria determine the number of changes of a substitution from one site to another (see [9]). In the remaining sections of this chapter, we only are interested in Fitch (or unweighted) parsimony for which all substitutions are given the same weight of one unit.

In the general case, the input of the problem is a multiple alignment comprising $n$ DNA sequences of length $m$ expressed over an alphabet $\Sigma$, where $\Sigma = \{-, \mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, ?\}$[1] consists of four nucleotides, the gap symbol –, and eventually, the missing character symbol: ?. The second input of the problem is a binary rooted or unrooted tree whose leaves are labeled with the sequences of $L$. Other nodes of the tree, called internal nodes, have two descendants (see Figure 26.1). We then can define two problems respectively called *small* and *large* parsimony problems.

**Definitio 26.1 (Small parsimony problem)** *Given a multiple alignment of length m of a set L of n sequences and a tree T whose leaves are labeled with sequences of L, find the parsimony score of T.*

To compute the overall cost (or score) of a tree (also known as *tree length*), Fitch's algorithm [11] gradually moves back from the leaves to the root and computes hypothetical ancestral taxa. This often is referred to as the first-pass of the algorithm (see Algorithm 26.4 for a more practical description). At each position of an internal node, a set of bases is assigned. If two descendants $x$ and $y$ of an internal node $v$ have

[1] It is also possible to use protein sequences with a 20-letter alphabet of amino acids.



(a) first-pass from leaves to root

(b) affection after second-pass

**Figure 26.2** First pass and affectation after second pass for a tree of score three under Fitch's optimality criterion.

some bases in common, then they are assigned to the internal node $L_v = L_x \cap L_y$. Otherwise, all bases of both descendants are assigned to the parent $L_v = L_x \cup L_y$, and a cost of one unit is added to the overall score of the tree (see Figure 26.2). The second pass of the algorithm, which starts from the root and reaches the leaves, enables researchers to assign one nucleotide for a site if many possibilities exist to obtain a hypothetical tree. However, only the first pass is sufficient to obtain the parsimony score.

**Definitio 26.2 (Large parsimony problem or maximum parsimony problem)** *Given a mulitple alignment of length m of a set L of n sequences, find the most parsimonious tree T (i.e., a tree with minimum parsimony score).*

The score of a tree can be computed in polynomial time [11]. A rooted binary tree of $n$ leaves has $n - 1$ internal nodes; thus, the complexity of the small parsimony problem is $O(n \times m)$. Indeed, we need to compute the hypothetical sequences of $n - 1$ internal nodes to obtain the score of parsimony. However, the search for an optimal tree is computationally intractable; the large parsimony problem is extremely difficult to solve because it is equivalent to the nondeterministor polynomial (NP)-complete Steiner problem in a hypercube [12]. This is why, as we shall see later on, heuristics methods constitute the main alternative to obtain near-optimal trees with reasonable computation time [21, 36].

## 26.3 METHODS

### 26.3.1 Combinatorial Optimization

A combinatorial optimization problems consist of finding the *best* object, also called the optimum or optimal solution $s^{\star}$, in a finite (or possibly countably infinite) set of objects called solutions. This class of problems is generally NP-complete [15], which roughly means that the computing time needed by an algorithm to find an optimal solution would increase exponentially with the size of the problem to solve. Because of the importance of these problems, many algorithms have been developed. These algorithms are of twofold:

- *Exact* algorithms are guaranteed to find the best solution but might need exponential computation time; they try to optimize the search by ignoring configurations that can be identified as inappropriate during the search.
- *Approximate* methods trade optimality for efficiency by examining an appropriate subset of the solutions to find a good near optimal solution.

### 26.3.2 Exact Approach

***26.3.2.1 Exhaustive Enumeration.*** The simplest algorithm that can be designed to find the most parsimonious tree(s) is to generate all possible trees and compute their parsimony scores. However, tree searches are extremely difficult because the number of possible trees grows exponentially with the number of taxa (see Table 26.1).

***26.3.2.2 Branch and Bound.*** A first alternative to tackle the complexity of MP is the branch and bound (B&B) algorithm [23]. We first generate a tree, not necessarily optimal, and compute its parsimony score, which will serve as an upper bound. We then start the construction of a new tree, initially empty, and add a new taxon at each iteration. Each new taxon is put on all possible branches of the previous trees and generates a set of new trees, which are put in a list. The trees that have a parsimony score greater than the upper bound are withdrawn from the list. The main drawback of the B&B algorithm is that the list of trees is too important for the algorithm to be efficient, which is why the algorithm only can be applied on a set of less than 20 taxa.

### 26.3.3 Local Search Methods

Because of the amount of trees to evaluate and the inefficiency of exact methods, it is preferable to use approximate approaches for inferring large phylogenetic trees.

**Table 26.1 Number of unrooted and rooted binary trees**

| Number of Taxa | Number of Unrooted Trees | Number of Rooted Trees |
|---|---|---|
| 10 | 2.0e + 06 | 3.4e + 07 |
| 20 | 2.2e + 20 | 8.2e + 21 |
| 30 | 8.6e + 36 | 4.9e + 38 |
| 40 | 1.3e + 55 | 1.0e + 57 |
| 50 | 2.8e + 74 | 2.7e + 76 |
| 80 | 2.1e + 137 | 3.4e + 139 |
| $n$ | $\prod_{i=3}^{n}(2i-5)$ | $\prod_{i=2}^{n}(2i-3)$ |

Local search (LS) uses iterative improvements to seek for solutions of better quality [25]. A LS algorithm consists of four essentials parts:

- A search space $S$ comprising a set of candidate solutions
- An evaluation function $f(s)$ of a solution $s \in S$, also called *fitness* function to assess the quality of a solution
- A neighborhood function $N(s) \subset S$ to define for each solution a subset of solutions that can be obtained by slightly modifying the current solution
- A transition strategy to accept or reject a neighboring solution

Typically, a LS algorithm (see algorithm 26.1) starts from an initial solution $s$ and then iteratively replaces the current solution by a neighbor $s' \in N(s)$ of better quality until no improving neighbor can be found. This process sometimes is called a *replication* in the MP literature and a *descent* for the optimization community. The replacement of the current solution favors neighbors of better quality with the intent to improve progressively the quality of the solution.

**Algorithm 26.1**

```
descent ($S, f, N$)
  $s$ := choose or generate an initial solution $\in S$
   for a given number of iterations $i$  do
    find $s' \in N(s)$ such that $f(s') < f(s)$ or  return $s$
    $s$ := $s'$
   end for
   return s
```

In the case of MP, the search space is the set of all possible binary (rooted or unrooted) trees, and the fitness function is the parsimony score of a tree. The search for the most parsimonious tree is then a *minimization* problem. A solution $s_1$ is better than $s_2$ if $f(s_1) < f(s_2)$, and the solution $s_2$ is said to be of lower quality than $s_1$.

***26.3.3.1 Generation of the Initial Solution.*** We can generate an initial solution by two means. We either can generate a solution by randomly selecting taxa that are put on any branch of the generated tree or use an approximate method called *stepwise addition*. The stepwise addition, also known as Wagner trees, is close to the B&B algorithm but only keeps track of one most parsimonious tree. Each new taxon is inserted on all possible branches but only the tree that gives the best score is retained. The order in which the taxa are added successively plays an important role and the method generally will produce suboptimal trees. Trees obtained by B&B generally provide final solutions of better quality than trees generated through a random process.

***26.3.3.2 The Local Optimum Problem.*** The main drawback of LS is that it often can get stuck in a *local optimum*, namely, a solution that is not the optimal solution $s^\star$ but that is locally better than its neighbors. More formally, a local optimum $s^\diamond$ is such that $\forall s' \in N(s^\diamond)$, $f(s^\diamond) \leq f(s')$ and $f(s^\diamond) > f(s^\star)$.

To escape from a local optimum, several techniques have been designed. Some techniques allow the selection of neighbors of same or lower quality than the current solution, whereas others modify the evaluation function or perturb the current solution. For example:

1. The *side-walk descent* allows us to choose improving or equivalent neighbors during a certain number of iterations contrary to a pure descent algorithm, which only accepts strictly improving neighbors; this less restrictive condition allows escape from a local optimum and provides more randomness to the process.
2. The *random walk* is a similar process that offers the possibility to accept deteriorating neighbors (*i.e.*, of lower quality) with a given probability.
3. The well-known *simulated annealing* method (see section 26.3.3.4) is a specific random walk with a nonconstant probability to accept neighbors of lower quality depending on the importance of the deterioration as well as the progress of the search.

The noising techniques can modify either the current solution or the fitness function for a given number of iterations. The modification of the current solution is known as the *iterated local search* (ILS) [29], whereas the modification of the fitness function applied to MP is known as the *parsimony ratchet* [36, 26]. When a local optimum $s^\diamond$ is reached, the ratchet noises the evaluation function; the weights of a proportion of the characters (10–15%) can be increased or some characters can be eliminated. A second descent is performed from $s^\diamond$ using the noising evaluation function $f'$. Actually, $s^\diamond$ is generally not a local optimum in $(S, f')$, so the configuration is improved in this new search space (but deteriorated if we consider the initial fitness function). The solution $s'$ obtained from a descent using $f'$ is the starting point of a new LS process with the use of the initial score function $f$. This process is repeated during a fixed number of iterations.

**26.3.3.3 Neighborhoods.** For the MP problem different neighborhoods have been conceived that are identified under the term *branch-swapping*. They greatly influence the search for the best solution and are briefly recalled here.

*26.3.3.3.1 Traditional Neighborhoods.* Three complementary neighborhoods are traditionally used in phylogenetic reconstruction—NNI, SPR, and TBR (see Figure 26.3). Depending on the context and the community, these acronyms denominate either the local search algorithm that uses the related neighborhood or only the neighborhood function.

- NNI [46] consists in swapping two subtrees that are separated by a branch. This is a small neighborhood because each tree of $n$ leaves has $(2n - 6)$ NNI neighbors [42] ($n - 3$ internal branches and two possible swaps for each branch). An extension to NNI has been proposed by [13, 14] into a parametric

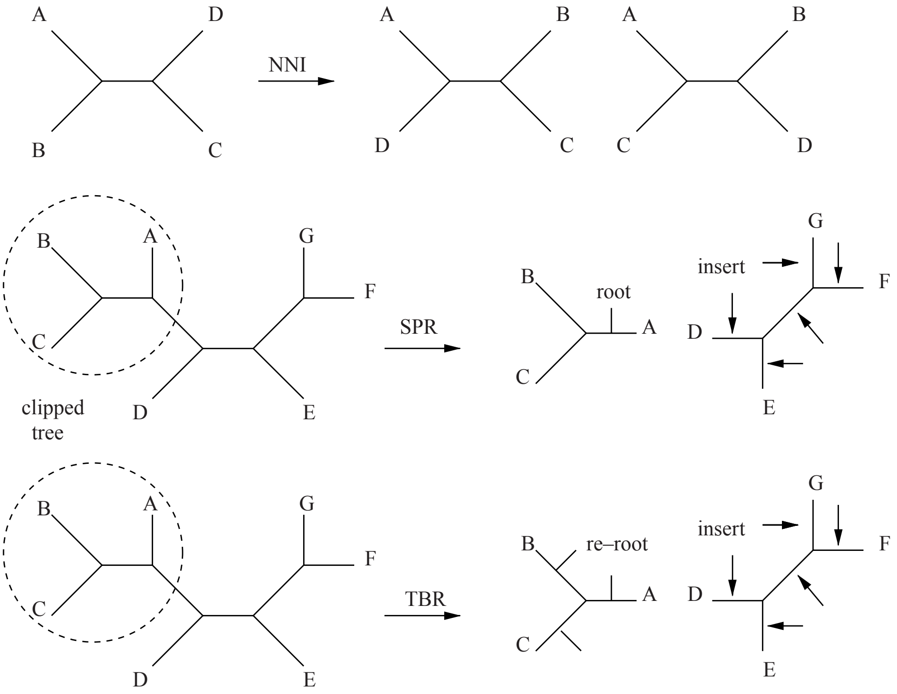


**Figure 26.3** Traditional neighborhoods for NNI, SPR, TBR.

neighborhood $p$-ECR (Edge-Contaction-and-Refinements), which shuffles $p$ adjacent branches. In particular, 1-ECR is equivalent to NNI.

- A *subtree pruning regrafting* (SPR) move [45] cuts a branch and creates two separate trees: the clipped tree and the residual tree. The clipped tree then can be regrafted on each branch of the residual tree to obtain a new topology. We can generate $2 \times (n - 3) \times (2n - 7)$ SPR rearrangements [1]. A particular case of SPR that moves only one leave in the tree is called STEP (Single Step).
- TBR (*Tree Bisection Reconnection*) [45] is a larger neighborhood that breaks the tree into two subtrees and reconnects the rerooted clipped tree to any branch of the residual tree. The number of TBR neighbors depends on the tree topology, but it is at least equal to $(2n - 3) \times (n - 3)^2$.

An important property is that these three neighborhoods are related: NNI $\subseteq$ SPR $\subseteq$ TBR, and have three distinct levels of complexity, respectively: $O(n)$, $O(n^2)$, and $O(n^3)$. At least one of these neighborhoods is used in every phylogenetic reconstruction software based on branch swapping.

The complexity of the LS algorithm thus mainly depends on the complexity of the neighborhood. For example, LS + SPR has a worst complexity of $O(i \times n^3 \times m)$: The complexity of the construction of a tree is $O(n \times m)$, and the complexity of SPR is $O(n^2)$. Finally, $i$ is the number of iterations of the LS algorithm.

*26.3.3.3.2 Variable Neighborhoods.* A neighborhood affects two main factors of an LS algorithm: the quality of the solutions found and the computation time. A small complexity neighborhood like NNI enables performing a quick search and is time scalable with the size of instances. When the number of taxa is increased, the size of the search space only grows linearly. The main drawback is that the current solution does not undergo enough modifications, and there is a high probability of observing a premature convergence to a local optimum of poor quality. For a large-size neighborhood like TBR, the number of neighbors to evaluate makes the search more computationally intensive, but the improvements can be important. It follows that SPR, as a medium-size neighborhood, often is used by descent algorithms as it permits obtaining solutions of better quality than NNI with less computation time than TBR.

Based on the observation that the size of a neighborhood influences the search, [32] has introduced the notion of *Variable Neighborhood Search* (VNS). The principle of the VNS metaheuristics is to use successively different neighborhoods during a descent by starting from a small-size neighborhood until the search is stuck in a local optimum; then one uses a neighborhood of larger size to allow important modifications of the current solution.

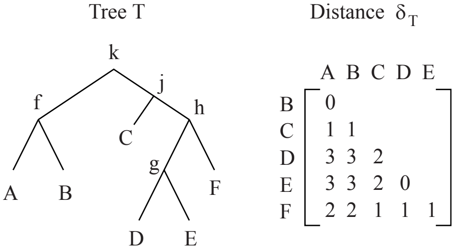
For example, an application of VNS to MP was proposed by Ribeiro and Vianna [40] with the use of two neighborhoods: SPR and 2-SPR (where $l$-SPR is the composition of $l$ SPR transformations). The algorithm starts with a SPR descent and switches to 2-SPR when a local optimum is found. They obtained good results despite an important computation time. In practice, $l$ does not exceed 2 because the $l$-SPR neighborhood with a size of $O(n^{2^l})$ rapidly becomes too large.

*26.3.3.3.3 Progressive Neighborhood.* Based on the observation that important topological modifications of the tree only are performed at the beginning of the descent, the authors of [19] have proposed a *progressive neighborhood* (PN), which contrary to VNS, starts with a medium size neighborhood (SPR) and is reduced iteratively to NNI. Results show that PN needs a smaller number of iterations than traditional SPR searches to obtain solutions of the same quality by limiting the evaluation of nonpertinent configurations. Recently, PN has been used by [37] to find consensus trees of high quality.

To make the neighborhood evolve, a topological distance on trees is defined in [19] that enables to building a distance matrix for a set of taxa given a tree topology. This distance also is used to control the size of the neighborhood (i.e., the distance between a pruned edge and its inserted edge is at most equal to a given limit).

**Definitio 26.3 (Topological distance)** *Let $i$ and $j$ be two taxa of a tree $T$. The topological distance $\delta_T(i, j)$ between $i$ and $j$ is defined as the number of edges of the path between parents of $i$ and $j$, minus one if the path contains the root of the tree.*

For example, on Figure 26.4, $A$ and $B$ have the same parent $f$, so $\delta_T(A, B) = 0$, and $\delta_T(A, D) = 3$ because the number of edges between $f$ and $g$ is 4 ($f \leftrightarrow k \leftrightarrow$



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline B & 0 & & & & \\ C & 1 & 1 & & & \\ D & 3 & 3 & 2 & & \\ E & 3 & 3 & 2 & 0 & \\ F & 2 & 2 & 1 & 1 & 1 \end{array}$$

**Figure 26.4** Example of topological distance $\delta_T$.

$j \leftrightarrow h \leftrightarrow g$), and as we pass through the root node $k$, we decrease the value by one unit. Note that for the topological distance, we consider trees as unrooted, which is why we remove one unit when passing through the root node. The progressive neighborhood based on the topological distance was implemented in the software Hydra [19]. The process used in Hydra to reduce the size of the neighborhood takes into account a parameter $M$, which corresponds to a maximum number of LS iterations. A parameter $d$ is introduced to control the size of the neighborhood and is defined as the maximal distance between a pruned edge and the edge where it is reinserted (*i.e.,* distance $\delta$ between their two descendant nodes). As such, changing $d$ leads to neighborhoods of different sizes that are explored with a descent algorithm.

***26.3.3.4 Other LS Algorithms.*** Several other LS algorithms can be used to solve the MP problem. Among them, we can distinguish the following:

*Tabu search* (TS) [17] is a kind of descent-ascendant method for which a neighbor can be chosen to replace the current solution even if it does not improve the fitness function. To avoid the problem of possible cycling and to allow the search to overcome the local optimum problem, TS introduces the notion of Tabu list—a short-term memory that maintains a selective history of previously encountered solutions. The size of the Tabu list $tt$, called Tabu tenure, prevents a solution to be reconsidered for the next $tt$ iterations. Yu-Min *et al.* [48] have applied TS to solve MP.

Simulated annealing (SA) [5, 27], like TS, accepts suboptimal solutions but with a certain probability. The principle of SA is inspired from annealing in metallurgy—a technique that involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. By analogy with this physical process, at each iteration, the current solution is replaced by a random neighbor chosen with a probability that depends on the difference between the fitness function value and a global parameter T (called the temperature). The temperature is decreased gradually during the process such that the current solution changes almost randomly at the beginning of the search, and only improving solutions are accepted toward the end of the search. The software LVB [4] is an application of SA to MP.

The metaheuristic greedy randomized adaptive search procedure (GRASP) [10, 38] can be considered a hybridization between a greedy construction method (here stepwise addition) and an LS mechanism. At every $p$ iteration, GRASP reconsiders previous choices by improving the current (and incomplete) tree by local search. If $p = 1$, then an LS operation follows each taxon insertion. Generally, these LS stages are relatively short, and a complete LS is done at the end of the procedure (when all taxa are inserted). Moreover, LS stages allow consideration of more randomness in the greedy part. Ribeiro and Vianna have applied a GRASP+VNS heuristic and obtained results of good quality on a set of benchmarks [2, 40].

### 26.3.4 Evolutionary Metaheuristics and Genetic Algorithms

Evolutionary algorithms (EA) [24, 31] represent another family of metaheuristics. Among them, *genetic algorithms* (GA) are based on a population of constant size of candidate solutions that undergo an evolution process with the use of operators named crossover, mutation, and selection. The aim of the population is to explore different interesting areas of the search space to diversify the search. The crossover operator aims to create new candidate solutions (offspring) by combining two or more solutions (parents). The mutation operator locally alters offspring by introducing randomness and consequently favoring diversity. Finally, the selection operator determines, which offspring will survive and reproduce. Different selection strategies exist like the roulette-wheel selection [3] or the tournament selection [20] for which the best individual is chosen after randomly picking $n$ individuals. If an offspring survives, then it replaces one of the individuals of the population.

**Algorithm 26.2**

```
Genetic-Algorithm ($S, f, N, x$)
  $P$ := { choose or generate $n$ individuals $\in S$ }
   for a given number of crossovers $x$  do
    $p, q$ :=  select-parents(P)
    $r$ :=  crossover($p, q$)          // r is the offspring of p and q
     mutation($r$)
     if  selection(r)  then
       replace($P, r$)
     end if
   end for
```

**26.3.4.1 GA Related Problems.** It is now widely acknowledged that genetic operators like crossover and mutation should be tailored to the target problem to integrate problem-specific constraints and thus improve the search.

The literature describes several evolutionary algorithms for phylogenetic reconstruction: for instance [28, 30], for the maximum likelihood problem [6, 7, 33, 39] for the MP problem, and [8] for distance-based phylogenetic approaches. Note that conventional subtree crossover operators used in tree-based genetic programming are not directly applicable here.

Most of these tree crossover operators follow the *subtree cutting and regrafting* strategy. More precisely, given two parents, a subtree first is selected from one parent (the donor). Then the leaves of this subtree are deleted from the other parent (the receiver), leading to an intermediate tree. The final child tree is obtained by regrafting the subtree from the donor on a merge point of the intermediate tree. Obviously,

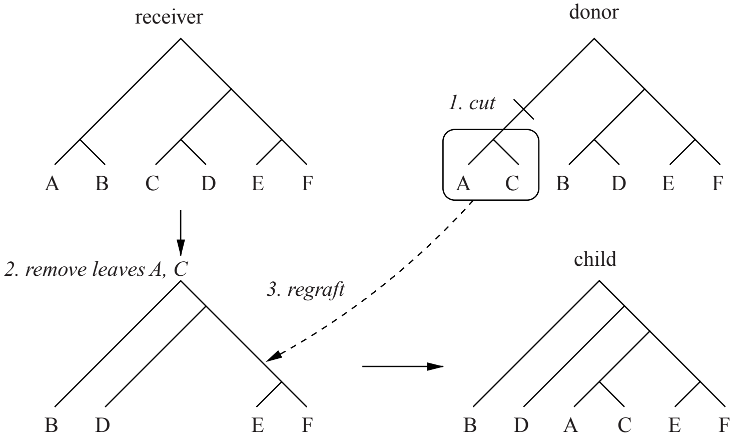


**Figure 26.5** Cut and regraft example.

exchanging the donor and receiver allows for obtaining a second child. Figure 26.5 shows an example with six taxa, where the subtree $(A, C)$ is removed from the donor and reinserted in the receiver.

One can observe that with such a crossover strategy, only partial information is transmitted from the parents to the offspring. In the example given in Figure 26.5, a subtree with two leaves (out of six) of the donor tree is passed on to the child. In one sense, only a small portion of information of the donor is transmitted, whereas a larger portion of information related to the four other taxa of the donor tree is lost during the crossover operation. To ensure a global combination and transmission of information during crossover operations, [18] introduces a specific operator based on the notion of *topological distance* between two leaves. The two parents are transformed into distance matrices, which are combined by using arithmetic operators to obtain an offspring matrix. From the offspring matrix, a tree is generated using a distance method (*e.g.*, unweighted pair group method with arithmetic mean [UPGMA] or neighbor joining [NJ]). This operator named distance-based information preservation crossover (DiBIP) will be described in the next section.

Tree-fusing [22] is another example of a crossover operator. Given two parent trees, this technique selects an improving tree among the population of valid trees constituted by exchanging subtrees between the parents.

Another crucial point to keep in mind is the importance of the diversification of the population. If the individuals are *too close*, then the search only will focus on one area of the search space. Generally, solutions of optimization problems are represented as vectors of values, but in the case of MP, the solutions are trees; although it is easy to compare vectors, it is less obvious to compare trees.

***26.3.4.2 Distance-Based Information Preservation Crossover.*** DiBIP crossover is based on the notion of *topological distance* (see Definition 26.3) and aims to preserves *common* properties of parents in terms of topological distance between taxa. By common, we mean that two taxa that are close (respectively, far) in both parents should stay close (respectively, far) in the child.

The general approach (see Algorithm 26.3) can be summarized as follows: (i) calculate a distance matrix for each parent tree, (ii) combine the matrices of the two parents to get a third matrix, and (iii) create a child tree from this new matrix. $T_1$ and $T_2$ represent the input trees (parents), $\Delta$ is a tree-to-distance operator that allows obtaining a distance matrix from a tree, $\oplus$ is a matrix operator that allows combining two distance matrices to produce a new distance matrix, and $\Lambda$ is a distance-to-tree operator that constructing a tree given a distance matrix. The three operators $\Delta$, $\oplus$, and $\Lambda$ represent the three steps to transform two parent trees into one child that preserves topological properties shared by both parents.

**Algorithm 26.3**

```
function DiBIP($T_1$, $T_2$, $\delta_T$, $\Delta$, $\oplus$, $\Lambda$)
$D_i$ := $\Delta(T_i)$ for $(i = 1, 2)$
$D^* := D_1 \oplus D_2$
$T^* := \Lambda(D^*)$
return $T^*$
```

This general scheme results in several comments; first, the distance measure ideally should be correlated to the evolutionary changes between taxa. For instance, two taxa separated by few evolutionary changes should have a smaller distance than two taxa separated by many changes. For example, the *Hamming distance* is not appropriate here because this metric is totally independent of tree topologies.

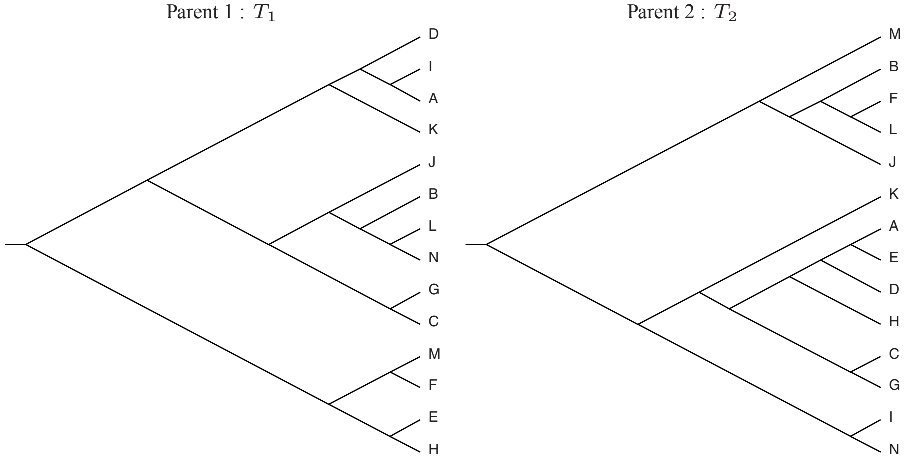
Second, to preserve common properties of the parents during the crossover operation, a valid matrix operator $\oplus$ should meet some specific requirements meaningful to the MP problem to help transmit properties shared by both parents to the child. For instance, if a pair of taxa $(A, B)$ is closer than another pair $(C, D)$ in both parents, then this property should be conserved by the crossover process and be transmitted to the resulting child. The arithmetic mean is a simple valid operator, even if it is possible to favor closeness or distant relations with the parametric operator given by $(D_1 \oplus D_2)(i, j) = \alpha . \min\{D_1(i, j), D_2(i, j)\} + (1 - \alpha). \max\{D_1(i, j), D_2(i, j)\}$, with $\alpha \in [0, 1]$.

Although the resulting distance matrix consists of topological distances and not evolutionary ones, a simple clustering algorithm like UPGMA is well-adapted to provide a child tree. It ensures aggregating taxa that are close in parents (depending on the selected $\oplus$ operator instantiation).

To illustrate this technique, we provide an example with two parents (see Figure 26.6) for which we compute topological matrices $D_1$ and $D_2$ (see Figure 26.7). For simplicity's sake, we use the addition operator to combine the two parent matrices into the child matrix $D^*$ (see Figure 26.8).

## 26.3.5 Memetic Methods

A memetic or hybrid algorithm (MA) is the combination of a GA helped by an LS improver [34]. Each time a new individual is generated by the GA, it is submitted


Parent 1 : $T_1$

Parent 2 : $T_2$

**Figure 26.6** Example of input trees $T_1$ and $T_2$ for the DiBIP crossover.

| $D_1$ | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | - | **B** | | | | | | | | | | | | |
| B | 6 | - | **C** | | | | | | | | | | | |
| C | 5 | 3 | - | **D** | | | | | | | | | | |
| D | 1 | 5 | 4 | - | **E** | | | | | | | | | |
| E | 5 | 5 | 4 | 4 | - | **F** | | | | | | | | |
| F | 5 | 5 | 4 | 4 | 2 | - | **G** | | | | | | | |
| G | 5 | 3 | 0 | 4 | 4 | 4 | - | **H** | | | | | | |
| H | 5 | 5 | 4 | 4 | 0 | 2 | 4 | - | **I** | | | | | |
| I | 0 | 6 | 5 | 1 | 5 | 5 | 5 | 5 | - | **J** | | | | |
| J | 5 | 1 | 2 | 4 | 4 | 4 | 2 | 4 | 5 | - | **K** | | | |
| K | 2 | 4 | 3 | 1 | 3 | 3 | 3 | 3 | 2 | 3 | - | **L** | | |
| L | 7 | 1 | 4 | 6 | 6 | 6 | 4 | 6 | 7 | 2 | 5 | - | **M** | |
| M | 5 | 5 | 4 | 4 | 2 | 0 | 4 | 2 | 5 | 4 | 3 | 6 | - | **N** |
| N | 7 | 1 | 4 | 6 | 6 | 6 | 4 | 6 | 7 | 2 | 5 | 0 | 6 | - |

| $D_2$ | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | - | **B** | | | | | | | | | | | | |
| B | 8 | - | **C** | | | | | | | | | | | |
| C | 4 | 6 | - | **D** | | | | | | | | | | |
| D | 1 | 7 | 3 | - | **E** | | | | | | | | | |
| E | 0 | 8 | 4 | 1 | - | **F** | | | | | | | | |
| F | 9 | 1 | 7 | 8 | 9 | - | **G** | | | | | | | |
| G | 4 | 6 | 0 | 3 | 4 | 7 | - | **H** | | | | | | |
| H | 2 | 6 | 2 | 1 | 2 | 7 | 2 | - | **I** | | | | | |
| I | 6 | 4 | 4 | 5 | 6 | 5 | 4 | 4 | - | **J** | | | | |
| J | 7 | 1 | 5 | 6 | 7 | 2 | 5 | 5 | 3 | - | **K** | | | |
| K | 4 | 4 | 2 | 3 | 4 | 5 | 2 | 2 | 2 | 3 | - | **L** | | |
| L | 9 | 1 | 7 | 8 | 9 | 0 | 7 | 7 | 5 | 2 | 5 | - | **M** | |
| M | 6 | 2 | 4 | 5 | 6 | 3 | 4 | 4 | 2 | 1 | 2 | 3 | - | **N** |
| N | 6 | 4 | 4 | 5 | 6 | 5 | 4 | 4 | 0 | 3 | 2 | 5 | 2 | - |

**Figure 26.7** Topological distance matrices of $T_1$ and $T_2$.

$D^* = D_1 + D_2$

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | - | **B** | | | | | | | | | | | | |
| B | 14 | - | **C** | | | | | | | | | | | |
| C | 9 | 9 | - | **D** | | | | | | | | | | |
| D | 2 | 12 | 7 | - | **E** | | | | | | | | | |
| E | 5 | 13 | 8 | 5 | - | **F** | | | | | | | | |
| F | 14 | 6 | 11 | 12 | 11 | - | **G** | | | | | | | |
| G | 9 | 9 | 0 | 7 | 8 | 11 | - | **H** | | | | | | |
| H | 7 | 11 | 6 | 5 | 2 | 9 | 6 | - | **I** | | | | | |
| I | 6 | 10 | 9 | 6 | 11 | 10 | 9 | 9 | - | **J** | | | | |
| J | 12 | 2 | 7 | 10 | 11 | 6 | 7 | 9 | 8 | - | **K** | | | |
| K | 6 | 8 | 5 | 4 | 7 | 8 | 5 | 5 | 4 | 6 | - | **L** | | |
| L | 16 | 2 | 11 | 14 | 15 | 6 | 11 | 13 | 12 | 4 | 10 | - | **M** | |
| M | 11 | 7 | 8 | 9 | 8 | 3 | 8 | 6 | 7 | 5 | 5 | 9 | - | **N** |
| N | 13 | 5 | 8 | 11 | 12 | 11 | 8 | 10 | 7 | 5 | 7 | 5 | 8 | - |

Child : $T^* = \Lambda(D^*)$

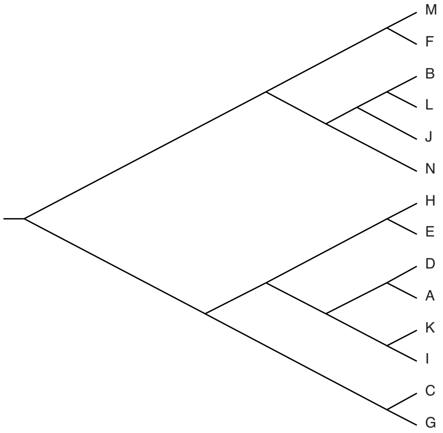


**Figure 26.8** Topological distance matrix of child and tree that results from UPGMA for the DiBIP crossover.

to an LS improvement. In Algorithm 26.2, the mutation then is followed by a descent that starts with $r$. MA alternates intensification (local search) and diversification (crossover) stages. Despite the fact that MA are more computationally intensive, they tend to provide solutions of better quality than GA. For example, Hydra [19] is an implementation of an MA.

### 26.3.6 Problem-Specific Improvements

Different techniques that are specific to MP have been developed to improve the search or decrease the computation time needed to obtain a tree of minimum score.

***26.3.6.1 Divide and Conquer Methods.*** For example, sectorial search (SS) [22] is a special kind of tree rearrangement that focuses on sectors (subtrees) of a tree. SS is based on the divide and conquer principle in which subtrees can be analyzed more quickly than the overall tree. Sectors are analyzed separately, and if a better configuration is found, then it will replace the previous one. Three different kinds of SS have been defined in [22]: *random* for which sectors are selected randomly, *consensus-based*, and *mixed*. Experiments show that a sector should have between 35 and 55 nodes to obtain significant improvements. Another family of divide-and-conquer methods are the disc-covering methods (DCM) [35, 44]: DCM1 can be considered an attempt to produce overlapping clusters to minimize the intracluster diameter and produce good subproblems. Nevertheless, the structure induced by the decomposition is often poor. DCM2 improves DCM1, but the subproblems obtained tend to be too large. Finally, DCM3 uses a dynamically updated guide tree to direct the decomposition. It then enables focusing the search on the *best* parts of the search space.

***26.3.6.2 Multi-Character Optimization.*** The most time-consuming function in a search algorithm is the function of Fitch (see Algorithm 26.4). This function takes as input two taxa $t1$ and $t2$. The output is the hypothetical taxon $t3$ and the number of changes observed that contributes to the parsimony score. Remember that the sum of all changes of the overall tree constitutes the parsimony score.

We can implement this function by taking full advantage of some relevant features offered by modern ×86 processors. More precisely, the core of modern ×86 processors has a SSE (single instruction multiple data [SIMD] streaming extension) unit that enables *vectorizing* the code. The vectorization of the code means applying the same operation on different data at the same time. Intel and AMD processors offer a set of 8 SSE registers of 128 bits long on a 32-bit architecture. If we represent a nucleotide with one byte, then a SSE register can store and handle 16 bytes (nucleotides) at a time.

To perform efficiently the union and intersection of Algorithm 26.4, each character is represented by a power of two, from $2^0 = 1$ (–) to $2^4 = 16$ $(T)$, except for `?`, which can represent any other character, and then is coded by the value $31 = 1 + 2 + \cdots + 16$. The union can be performed by the binary-OR ( | ) and the intersection by the binary-AND (&). The vectorization of Fitch's function gives a 90% improvement on Intel Core 2 Duo processors, whereas other architectures (Pentium II/III/4, Pentium-M, Athlon 64, and Sempron) provide 70–80% improvement. This improvement then enables dividing the overall computation time of a program by a factor of three to four. A first pseudocode was given in [43] for PowerPC processors and recently [41] was released the code for Intel and AMD processors. Finally, note that the most recent processors (Intel Core i5 or i7 and AMD Phenom) introduce the SSE4.2 instruction set that contains the POPCNT (POPulation CouNT) instruction, which counts the number of bits set to one in a general purpose register. This instruction is used essentially to determine the number of changes that occur when one performs the union between `x[i:i+15]` and `y[i:i+15]`. By replacing the implementation of POPCNT by the native SSE4.2 instruction, the experiments we have carried out show an overall improvement of 95% (on an Intel Core i7 860 processor) compared with the basic implementation.

**Algorithm 26.4**

```
function fitch($v, x, y$ : array[1..m] of $character$) : $integer$
  $changes := 0$
   for i := 1  to m  do
    $v[i] := x[i] \cap y[i]$                                   // $L_v = L_x \cap L_y$
     if ($v[i] == 0$)  then
      $v[i] := x[i] \cup y[i]$                                 // $L_v = L_x \cup L_y$
      $changes := changes + 1$
     end if
   end for
   return $changes$
```

***26.3.6.3 Fast Character Optimization Techniques.*** A set of methods [21, 16, 43, 47] fall into the category of *fast character optimization* techniques (*i.e.,* a set of shortcuts that helps decrease the computation time by not recalculating the whole tree each time a SPR or TBR modification is applied). Those techniques are particularly effective when an important number of SPR or TBR neighbors has to be evaluated. For Fitch's parsimony, characters are considered as unordered and multi-state; they can transform from one state to another independently. As a consequence, an unrooted tree may be rooted on any branch with no modification of the parsimony score, which means there is a potential root node for any branch.

In [21], Goloboff proposed a method for the indirect calculation of the parsimony score which uses two passes. This method needs only to compare the root of the clipped tree with the potential root of the target tree to obtain the score of a potential new tree for a SPR search. Gladstein [16] also proposed an algorithm that is exact and correct. In [47], a two-pass algorithm is described that has the same complexity of Goloboff's and is faster than the incremental method of Gladstein.

## 26.4 CONCLUSION

Because of its inherent complex structure, the resolution of the large maximum parsimony problem can be achieved efficiently by means of optimization techniques.

Table 26.2 gives an overview of the complexity of the different methods described throughout this chapter, where $n$ represents the number of taxa and $m$ represents the number of sites of each taxa. For local search, $i$ represents the number of iterations of the search. For genetic algorithms, $p$ is the size of the population, $X$ is the number of crossovers and cross, mut, sel are the complexity of the crossover, mutation, and selection operations, respectively. The last term LS is the complexity of the LS method used to improve a solution.

Local search methods are the standard resolvers for the maximum parsimony problem and are used widely. Genetic algorithms can obtain results of better quality than LS only if the crossover, mutation, and selection operators are tailored to the problem. Finally, memetic algorithms, as a combination of GA and LS, are the methods that can achieve results of very good quality, but they are often more time consuming compared to an LS. A lot of other techniques, like the ratcheting technique

**Table 26.2 Complexity of methods**

| Method | Complexity |
|---|---|
| Exhaustive | $O(n^n \times m)$ |
| Branch and bound | $O(n^n \times m)$ |
| Stepwise | $O(n^3 \times m)$ or $O(n^4 \times m)$ |
| LS+NNI | $O(i \times n^2 \times m)$ |
| LS+SPR | $O(i \times n^3 \times m)$ |
| LS+TBR | $O(i \times n^4 \times m)$ |
| GA | $O(p \times n \times m) + X \times (\text{cross} + \text{mut} + \text{sel})$ |
| MA | $O(p \times n \times m) + X \times (\text{cross} + \text{mut} + \text{sel} + \text{LS})$ |

or sectorial search, are also very useful to escape from local optimum or to improve the overall tree locally.

New tools like the topological distance and the DiBiP crossover show that new approaches can help design *clever* techniques to help solve the MP problem more effectively and to reach solutions of higher quality.

## REFERENCES

1. B.L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Ann Combinator*, 5(1):1–15, 2001.
2. A.A. Andreatta and C.C. Ribeiro. Heuristics for the phylogeny problem. *JHEU*, 8:429–447, 2002.
3. J.E. Baker. Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms and their Application (ICGA2)*, 1987, pp. 14–21.
4. D. Barker. Parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, 20:274–275, 2004.
5. V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *J Optim Theory Appl*, 45:41–51, 1985.
6. C.B. Congdon. Gaphyl: An evolutionary algorithms approach for the study of natural evolution. *Proceedings of the 6th Joint Conference on Information Science*, 2002.
7. C.B. Congdon and K.J. Septor. Phylogenetic trees using evolutionary search: Initial progress in extending gaphyl to work with genetic data. *Proceedings of the 2003 Congress on Evolutionary Computation*, 2003, pp. 320–326.
8. C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. *Lect Notes Comput Sci*, 2439:720–729, 2002.
9. J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2003.
10. T.A. Feo and M.G.C. Resende. Greedy randomized adaptative search procedures. *J Global Optim*, 6:109–133, 1995.
11. W. Fitch. Towards defining course of evolution: Minimum change for a specified tree topology. *Syst Zool*, 20:406–416, 1971.

12. L.R. Foulds and R.L. Graham. The steiner problem in phylogeny is np-complete. *Adv Appl Math*, 3:43–49, 1982.
13. V. Ramachandran, G. Ganapathy, and T. Warnow. Better hill-climbing searches for parsimony. *Proceedings of the Third International Workshop on Algorithms in Bioinformatics (WABI)*, 2003, pp. 245–258.
14. V. Ramachandran, G. Ganapathy, and T. Warnow. On contract-and-refine transformations between phylogenetic trees. *SODA*, 2004, pp. 900–909.
15. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.
16. D.S. Gladstein. Efficient character optimization. *Cladistics*, 13:21–26, 1997.
17. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, Boston, MA, 1997.
18. A. Goëffon, J.-M. Richer, and J.K. Hao. A distance-based information preservation tree crossover for the maximum parsimony problem. *Lect Notes Comput Sci*, 4193:761–770, 2006.
19. A. Goëffon, J.-M. Richer, and J.K. Hao. Progressive tree neighborhood applied to the maximum parsimony problem. *IEEE/ACM Trans Comput Biol Bioinform*, 5(1):136–145, 2008.
20. D.E. Goldberg and K. Deb. *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*. Morgan Kaufmann, New York, 1991, pp. 69–93.
21. P.A. Goloboff. Character optimization and calculation of tree lengths. *Cladistics*, 9:433–436, 1993.
22. P.A. Goloboff. Analyzing large data sets in reasonable times: Solutions for composite optima. *Cladistics*, 15:415–428, 1999.
23. M.D. Hendy and D. Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Math Biosci*, 59:277–290, 1982.
24. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
25. H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, New York, 2004.
26. I. Horovitz. A report on one day symposium on numerical cladistics. *Cladistics*, 15:177–182, 1999.
27. S. Kirkpatrick, C. Gellat, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
28. P.O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Mol Biol Evol*, 15(3):277–283, 1998.
29. H.R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In: F. Glover and G. Kochenberger (Eds). *Handbook of Metaheuristics*. Kluwer Academic, Boston, MA, 2002.
30. H. Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. *Pacific Symposium on Biocomputing*, 1996, pp. 512–536.
31. M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998.
32. N. Mladenović and N.P. Hansen. Variable neighborhood search. *Comput Oper Res*, 24:1097–1100, 1997.
33. A. Moilanen. Searching for most parsimonious trees with simulated evolutionary optimization. *Cladistics*, 15(3):39–50, 1998.

34. P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, McGraw-Hill, Maidenhead, Berkshire, England, UK, 1999, 219–234.
35. L. Nakhleh, U. Roshan, K. St John, J. Sun, and T. Warnow. Designing fast converging phylogenetic methods. *Bioinformatics Suppl*, 17:190–198, 2001.
36. K.C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15:407–414, 1999.
37. S. Pirkwieser and G.R. Raidl. Finding consensus trees by evolutionary, variable neighborhood search, and hybrid algorithms. *Proceedings of Generic and Evolutionary Computation Conference (GECCO'08)*, 2008.
38. M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptative search procedures. In F. Glover and G. Kochenberger editors, *Handbook of Metaheuristics*. Kluwer Academic, Boston, MA, 2003.
39. C.C. Ribeiro and D.S. Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. *Proceedings of 2nd Bresil Workshop on Bioinformatics*, 2003, pp. 97–102.
40. C.C. Ribeiro and D.S. Vianna. A grasp/vnd heuristic for the phylogeny problem using a new neighborhood structure. *Int Trans Oper Res*, 12:1–14, 2005.
41. J.-M. Richer. Three new techniques to improve phylogenetic reconstruction with maximum parsimony. Technical Report, LERIA, 2008.
42. D.F. Robinson. Comparison of labeled trees with valency three. *J Combin Theor*, 11:105–119, 1971.
43. F. Ronquist. Fast fitch-parsimony algorithms for large data sets. *Cladistics*, 14:387–400, 2000.
44. U. Roshan, B.M.E. Moret, T.L. Williams, and T. Warnow. Rec-i-dcm3: A fast algorithmic technique for reconstructing large phylogenetic trees. *Proceedins of IEEE Computational Systems Bioinformatics Conference (CSB 04)*, 2004, pp. 98–109.
45. D.L. Swofford and G.J. Olsen. Phylogeny reconstruction. In: D. M. Hillis and G. Moritz (eds). *Molecular Systematics*, Sinauer Associates, 1990, 411–501.
46. M.S. Waterman and T.F. Smith. On the similarity of dendograms. *J Theor Biol*, 73:789–800, 1978.
47. M. Yan and D.A. Bader. Fast character optimization in parsimony phylogeny reconstruction. Technical Report TR-CS-2003-53, University of New Mexico, Albuquerque, NM, 2003.
48. L. Yu-Min, F. Shu-Cherng, and T.L. Jeffrey. A tabu search algorithm for maximum parsimony phylogeny inference. *Eur J Oper Res*, 176(3):1908–1917, 2007.

# 27

# MAXIMUM ENTROPY METHOD FOR COMPOSITION VECTOR METHOD

Raymond H.-F. Chan, Roger W. Wang, and Jeff C.-F. Wong

## 27.1 INTRODUCTION

In the past few decades, a large volume of molecular sequences has been collected, from which the evolution and traits of the related living organisms are investigated. These sequences all look simple; for instance, the DNA sequence, no matter how long it is, contains only four different nucleotides A, C, G, and T, so it is not surprising that on the surface, these sequences themselves cannot tell us much. To reveal the hidden information, the use of the so-called sequence comparison is an essential tool. Sequence comparison methods can be divided into two main categories: alignment-based [15, 17, 36, 37, 42, 50] and alignment-free [25, 28, 40, 43, 52].

The alignment-based methods use the dynamic programming (DP) method to "align" the sequences and then find the similarity and dissimilarity after the alignment. To compare two sequences of length $n$ by any alignment-based method, both the computational cost and the memory requirement are $\mathcal{O}(n^2)$ [54]. Because of the accuracy of the DP method, the alignment-based methods are used widely for analyzing gene sequences. However, different gene sequences may give different evolutionary results. For instance, based on the 16rRNA sequences, birds, which more closely are related to crocodilians, were grouped with mammals [56]. In addition,

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

based on the gene sequences for MHG-CoA reductase, Archaeoglobus fulgidus—a definite archaean—was assigned into the bacteria group [14]. Today, with the advent of sequence techniques, whole genome sequences generally have been accepted as excellent tools for the study of species differences and of evolution [16]. However, aligning the whole genomes is a very challenging problem because every species has its own gene content and gene order, and we do not know which two genes can be truly aligned. Furthermore, as the length of the genome sequences are usually very long, it is impossible to align the genome sequences because of the cost of the computational time and the memory requirement.

The alignment-free methods, in turn, are developed for overcoming the difficulty of the analysis of the whole genome phylogeny. They can be divided into three classes:

- The gene content method [43]
- The Data compression method [24]
- The composition vector (CV) method [25, 40]

For the gene content method, the distance between two species is defined by their number of common genes divided by the total number of genes in the genome sequences. The data compression method uses the distance between the compressed information from the genome sequences as the distance between the species. For the CV method, the composition vector first is constructed for each species based on its whole genome sequence, and the distance between the composition vectors is used as the distance between species. In this chapter, we only will shed some light on the CV method.

The CV method was proposed by Hao *et al.* [25] for the phylogeny of bacteria, and it was very successful. The CV method generally consists of four steps as follows:

1. Construct the frequency vectors — Different methods for constructing the frequency vectors are discussed based on the different types of biological sequences that are input.
2. Construct the composition vectors — The composition vectors are constructed with each entry being the signal-to-noise ratio. Several kinds of models are introduced for estimating the noise.
3. Compute the distance between composition vectors — Several distance measures are introduced and analyzed.
4. Build the phylogenetic trees — We use the neighbor-joining method to draw the phylogenetic trees.

As we will see below, there is a link between the maximum entropy optimization and some existing denoising formulas. Maximum entropy is being used increasingly as a general and powerful technique for making the classification of species through the biological sequences from noise itself when the data in the signal is obscured by noise and bias (*e.g.*, [26]). Entropy can be justified in information–theoretic terms. Not only will we present some denoising formulas and suggest which one is optimal but we also will introduce several models for the CV method and show that the CV method also can be applied successfully for phylogenetic analysis of tetrapod, *hepatitis B* virus (HBV), mammal, and chorophlast. We even show that the CV method can provide some reasonable results in which the alignment methods failed (see Example 1).

This chapter is divided into four sections. Section 27.1 gives the introduction. Section 27.2 includes the general formulation of the CV method. Section 27.3 presents the results using the CV method with different denoising formulas and compares them with other existing results. Section 27.4 gives the concluding remarks.

## 27.2 MODELS AND ENTROPY OPTIMIZATION

In Section 27.2.1, a list of formal definitions for the biological terms is introduced. In Section 27.2.2, two of the most common denoising formulas in literature are revisited—those advocated by Hao *et al.*'s formula [25, 40] and Yu *et al.*'s formula [58]. In particular, under the framework of the constrained optimization problem with the maximum entropy approach, we provide three new denoising formulas by means of the CV method (*cf.* (27.14), (27.16), and (27.17)). Based on the angle-based distance approach, various types of distance formulas also are introduced in Section 27.2.3. Phylogenetic tree construction is described in Section 27.2.4.

### 27.2.1 Definitions

**Definitio 27.1** *Consider a molecular sequence (DNA/RNA sequence or peptide/amino acid sequence) of length $N$. Any consecutive $k$ molecules within the sequence are called a $k$-string, where $1 \leq k \leq N$.*

**Definitio 27.2** *The observed frequency $f(\alpha_1\alpha_2\cdots\alpha_k)$ of a $k$-string $\alpha_1\alpha_2\cdots\alpha_k$ is defined as*

$$f(\alpha_1\alpha_2\cdots\alpha_k) = \frac{g(\alpha_1\alpha_2\cdots\alpha_k)}{N-k+1} \tag{27.1}$$

*where $g(\alpha_1\alpha_2\cdots\alpha_k)$ is the number of times that $\alpha_1\alpha_2\cdots\alpha_k$ appears in the sequence.*

Let us define the frequency vector for the gene sequence and genome sequence, respectively.

**Definitio 27.3** *For a gene sequence, whether a DNA sequence or a RNA sequence, there are $4^k$ possible $k$-strings. A vector is constructed with the frequency defined in (27.1) for each entry and is called the frequency vector.*

Consider the following nucleotide sequence that consists of A, C, G, and T such that

GACTACTACT

Set $k = 3$ and $N - k + 1 = 8$. The total number of possible different 3-string sequences is then $4^3$, and the frequency vector is given as follows:

$$\begin{bmatrix} f(\mathrm{AAA}) \\ f(\mathrm{AAC}) \\ \vdots \\ f(\mathrm{ACT}) \\ \vdots \\ f(\mathrm{CTA}) \\ \vdots \\ f(\mathrm{GAC}) \\ \vdots \\ f(\mathrm{TAC}) \\ \vdots \\ f(\mathrm{TTG}) \\ f(\mathrm{TTT}) \end{bmatrix}_{4^3} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 3/8 \\ \vdots \\ 2/8 \\ \vdots \\ 1/8 \\ \vdots \\ 2/8 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

It is worth mentioning that a window of the length of $k$-string is used that slides it through the sequences by shifting one window at a time to look for the frequencies of each $k$-string.

There are three kinds of sequences available from the whole genome sequence:

1. **The whole DNA sequence** — For the whole DNA sequence, the frequency of appearance of a $k$-string also is defined in (27.1).
2. **The protein-coding DNA sequences**

**Definitio 27.4** *For the protein-coding DNA sequences, the observed frequency of a k-string $\alpha_1\alpha_2\cdots\alpha_k$ in the whole sequence is defined as* [40]

$$f(\alpha_1\alpha_2\cdots\alpha_k) = \frac{\sum_{j=1}^{m} g_j(\alpha_1\alpha_2\cdots\alpha_k)}{\sum_{j=1}^{m}(N_j - k + 1)} \tag{27.2}$$

*where m is the number of protein-coding gene sequences from the whole genome, $g_j(\alpha_1\alpha_2\cdots\alpha_k)$ is the number of times that $\alpha_1\alpha_2\cdots\alpha_k$ appears in the*

*$j$th DNA sequence, and $N_j$ is the length of the $j$th DNA sequence. A frequency vector then is constructed with each entry containing all frequencies defined in* (27.2).

3. **The amino acid sequences of all protein-coding sequences** — For the amino acid sequences of all protein-coding sequences, the frequency vector can be constructed similarly, with each entry defined in (27.2). A vector of length $20^k$ then is constructed.

***27.2.1.1 Signal-to-Noise Ratio.*** It generally is accepted that the phylogenetic signals in the biological sequence data often are obscured by noise and bias [10]. The relation between the signal and the noise can be formulated as a single mathematical formula, referred to as the composition vector. Given $M$ molecular sequences, $M$ frequency vectors of the same length $|\Omega|^k$ were defined earlier, where

$$|\Omega| = \begin{cases} 4, & \text{if the sequence is the DNA/RNA type} \\ 20, & \text{if the sequence is the peptide/amino acid type} \end{cases} \tag{27.3}$$

**Definitio 27.5** *For each $f(\alpha_1\alpha_2\cdots\alpha_k)$, the frequency of appearance of the $k$-string $\alpha_1\alpha_2\cdots\alpha_k$ defined in* (27.1)*, we will estimate its noise and denote it by $q(\alpha_1\alpha_2\cdots\alpha_k)$. Then the composition vector of one species is the $|\Omega|^k$-vector, where each nonzero entry equals*

$$\frac{f(\alpha_1\alpha_2\cdots\alpha_k) - q(\alpha_1\alpha_2\cdots\alpha_k)}{q(\alpha_1\alpha_2\cdots\alpha_k)}$$

*the signal-to-noise ratio of the $k$-string $\alpha_1\alpha_2\cdots\alpha_k$.*

### 27.2.2 Denoising Formulas

Let us review some existing denoising formulas for removing noises in the phylogenetic signals.

***27.2.2.1 Hao* et al.*'s Formula.*** Given any molecular sequence, Hao *et al.* [25, 40] employed the following formula:

$$q^{\mathrm{Hao}}(\alpha_1\alpha_2\cdots\alpha_k) = \begin{cases} \dfrac{f(\alpha_1\cdots\alpha_{k-1})f(\alpha_2\cdots\alpha_k)}{f(\alpha_2\cdots\alpha_{k-1})}, & \text{if } f(\alpha_2\cdots\alpha_{k-1}) \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{27.4}$$

to estimate the noise of the $k$-string $\alpha_1\cdots\alpha_k$, where $f(\mathrm{u})$ is the frequency of appearance of any string u in the sequence. To find the noise of the $k$-string by (27.4), using the $(k-2)$th order Markov assumption (together with the joint and conditional probability) [34], the appearance frequencies of the $(k-1)$ string and the $(k-2)$ string are established. If the denominator in (27.4) $f(\alpha_2\cdots\alpha_{k-1})$ is zero, then it means that the $(k-2)$ string does not appear in the sequence. Obviously the $(k-1)$ strings $\alpha_1 \cdots \alpha_{k-1}$ and $\alpha_2 \cdots \alpha_k$ also will not appear in the sequence, and then

$$f(\alpha_1 \cdots \alpha_{k-1}) = f(\alpha_2 \cdots \alpha_k) = 0$$

When this degeneracy case happens, one simply can let

$$q^{\mathrm{Hao}}(\alpha_1 \cdots \alpha_k) = 0$$

Formula (27.4) is derived from the observed frequency $f(\cdot)$ that represents the probability. It was Brendel *et al.* [4] in 1986 who originally introduced (27.4) for revealing the functional and evolutionary relatedness of word sequence. Hao *et al.* [25, 40] used formula (27.4) for the phylogenetic analysis of prokaryotes based on whole genome sequences.

***27.2.2.2 Yu's Formula.*** Given any molecular sequence, Yu *et al.* [58] proposed the following formula:

$$q^{\mathrm{Yu}}(\alpha_1\alpha_2 \cdots \alpha_k) = \frac{f(\alpha_1)f(\alpha_2 \cdots \alpha_k) + f(\alpha_1 \cdots \alpha_{k-1})f(\alpha_k)}{2} \tag{27.5}$$

to find the noise of the $k$-string $\alpha_1\alpha_2 \cdots \alpha_k$, where $f(\mathrm{u})$ is the appearance frequency of any string u in the sequence. A salient feature of (27.5) is that it takes the average of the sum of two independent events with respect to $f(\alpha_1)f(\alpha_2 \cdots \alpha_k)$ and $f(\alpha_1 \cdots \alpha_{k-1})f(\alpha_k)$.

Formula (27.5) is taken from the observed frequency $f(\cdot)$ that represents the probability. Application of (27.5) was common in the area of complex and dynamic systems (*e.g.*, [57]). Yu *et al.* [58] used formula (27.5) for the phylogenetic analysis of prokaryotes, chloroplasts, and other phylogenetic problems based on whole genome sequences.

***27.2.2.3 Establishing Denoising Formulas Using the Maximum Entropy Principle.*** For the sake of simplicity, we only consider DNA/RNA sequences in our formulation, but the amino acid sequences can be used in a similar fashion.

Let us consider the following constraints [26]:

$$\begin{cases} q(\mathrm{vA}) + q(\mathrm{vC}) + q(\mathrm{vG}) + q(\mathrm{vT}) = f(\mathrm{v}) \\ q(\mathrm{Av}) + q(\mathrm{Cv}) + q(\mathrm{Gv}) + q(\mathrm{Tv}) = f(\mathrm{v}) \end{cases} \tag{27.6}$$

where $q(\cdot)$ is the frequency to be maximized from the entropy when the observed frequency $f(\mathrm{v})$ for all $(k-1)$ strings v are given. The solution of the optimization problem, is (27.4). We assume that the noises of the $k$-strings are related to (27.6) (*i.e.*, $q(\mathrm{vA}) + q(\mathrm{vC}) + q(\mathrm{vG}) + q(\mathrm{vT})$ and $q(\mathrm{Av}) + q(\mathrm{Cv}) + q(\mathrm{Gv}) + q(\mathrm{Tv})$

are known functions of v), and we assume that the two sums are not identical to each other because their values can be changed and will lead to different denoising formulas as we will see later on.

**_27.2.2.4 Formulation of the Optimization Problem._** Let us propose our noise model as follows: The noise $q(\cdot)$ of the $4^k$'s $k$-strings satisfies

$$\begin{cases} q(\text{vA}) + q(\text{vC}) + q(\text{vG}) + q(\text{vT}) = l(\text{v}) \\ q(\text{Av}) + q(\text{Cv}) + q(\text{Gv}) + q(\text{Tv}) = r(\text{v}) \end{cases} \tag{27.7}$$

where $l(\text{v})$ and $r(\text{v})$ are given nonnegative numbers for each $(k-1)$ string v, and the right-hand sides of (27.7) are obtained from the observed frequencies of any given sequence. Note that in (27.7), depending on the choice of $(k-1)$ strings, there are $(2 \times 4^{k-1})$ constraints and $4^k$ unknowns. Thus, when the number of constraints is fewer than the number of unknowns, the system is underdetermined, and the solution is not unique.

To obtain the unique $q(\text{u})$, we maximize their entropy. More precisely, let $q_i \equiv q(\text{u}_i)$ be the noise of the $k$-string $\text{u}_i$; then we obtain $q_i$ by solving the following constrained maximisation problem:

$$\begin{aligned} &\text{maximize } -\sum_{i=1}^{4^k} q_i \log q_i \\ &\text{subject to } \begin{cases} q_i \text{ satisfies (27.7)} \\ q_i \geq 0 \text{ for all } i \end{cases} \end{aligned} \tag{27.8}$$

We note that $-q_i \log q_i$ is the entropy of $q_i$.

**_27.2.2.5 Solution of the Optimization Problem._** According to Pevzner [39], the best $k$-string for a sequence of length $N$ is $\log_4 \left[\frac{N(N-1)}{2}\right]$. Thus, if $N = 1000$, then the best $k$-string is about 10. Hence the optimization problem (27.8) will have about 1 million unknowns, and it is seemingly difficult to solve such a constraint problem. However, we have the following useful result.

**Lemma 27.1** *For $k \geq 2$, the problem* (27.8) *is decoupled into* $4^{k-2}$ *subproblems of size 8-by-16 each.*

*Proof:*

- Let us first see the structure/pattern of the coefficient matrix in (27.8) when $k = 3$. The other choice of $k$ can be used similarly. As shown in Figure 27.1, the matrix is sparse and binary (containing 0 and 1 only), and the nonzero entries can be divided into two categories; the nonzero entries located on the

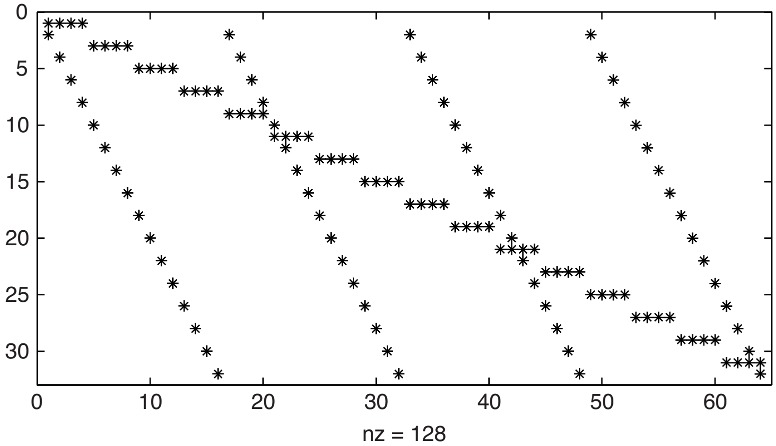


**Figure 27.1** The pattern of the coefficient matrix in (27.8).

"diagonal part" of the matrix form one category, whereas the rest of the nonzero entries form the other category. To decouple these two categories, one simply rearranges the order of the equations:

– Put the original odd-order equations first

– Locate the even-order equations later

The pattern of the new coefficient matrix is shown in Figure 27.2.

- In Figure 27.2, there are only four unknown variables $q_1$, $q_2$, $q_3$, and $q_4$ in the first row, and these four variables also are contained, respectively, in four other

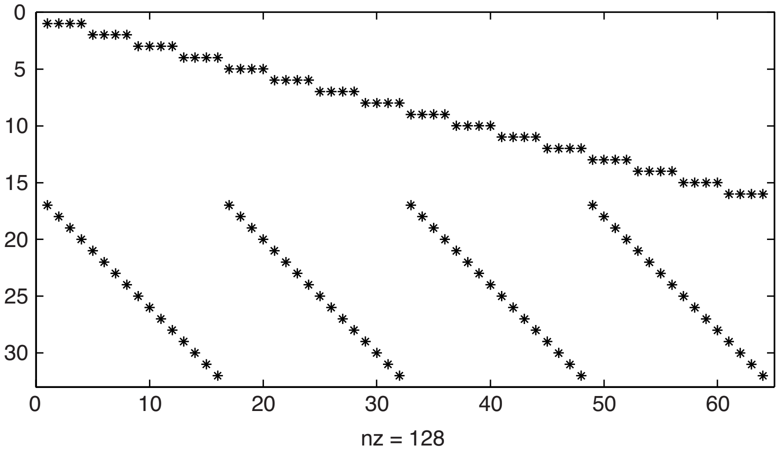


**Figure 27.2** The pattern of the coefficient matrix after the permutation.

rows of the matrix: rows 17–20. After carefully examining these four rows, we find that the following 16 variables

$$q_{16*(i-1)+j}, \qquad \forall\, i, j = 1, 2, 3, 4$$

will not be found anywhere else but totally are contained in the following eight constraints, given in ascending order:

$$\text{Constraint}: \; 1,\ 5,\ 9,\ 13,\ 17,\ 18,\ 19,\ 20$$

These eight constraints clearly are divorced from other constraints. Moreover, the 3-strings are contained in these eight constraints if and only if they can be written in the following form

$$\text{LAR}, \quad \forall\, \text{L}, \text{R} \in \{\text{A}, \text{C}, \text{G}, \text{T}\}$$

Similarly, three groups of eight constraints are formed if and only if we also have the following cases:

$$\begin{array}{ll} \text{LCR}, & \forall\, \text{L}, \text{R} \in \{\text{A}, \text{C}, \text{G}, \text{T}\} \\ \text{LGR}, & \forall\, \text{L}, \text{R} \in \{\text{A}, \text{C}, \text{G}, \text{T}\} \\ \text{LTR}, & \forall\, \text{L}, \text{R} \in \{\text{A}, \text{C}, \text{G}, \text{T}\} \end{array}$$

Now we conclude that the original system can be decomposed into four subsystems (see Figure 27.3), and the 3-strings are contained in each subsystem if and only if the 3-strings can be written in any of the four forms.

With this idea, we now consider the general formulation of the equality constraints when $k \geq 3$. Let us rewrite the $k$-strings in the left-hand side of (27.7)

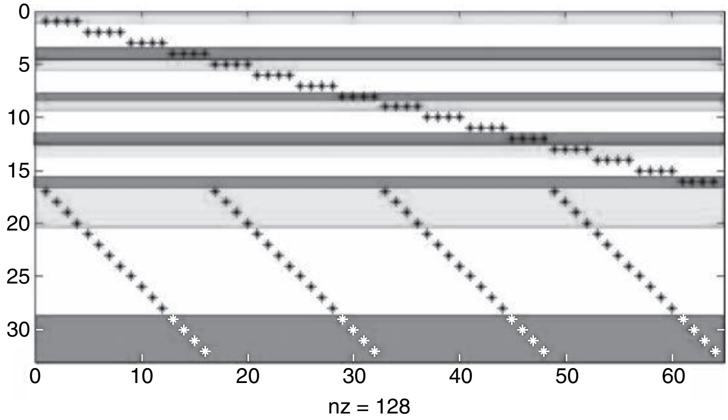


**Figure 27.3** The figure for the decoupling of the permuted coefficient matrix.

as LwR, where w is a $(k-2)$ string. By exhausting different combinations of L and R, we obtain a system of the following constraints for each $w$:

$$\begin{cases} q(\mathrm{AwA}) + q(\mathrm{AwC}) + q(\mathrm{AwG}) + q(\mathrm{AwT}) = l(\mathrm{Aw}) \\ q(\mathrm{CwA}) + q(\mathrm{CwC}) + q(\mathrm{CwG}) + q(\mathrm{CwT}) = l(\mathrm{Cw}) \\ q(\mathrm{GwA}) + q(\mathrm{GwC}) + q(\mathrm{GwG}) + q(\mathrm{GwT}) = l(\mathrm{Gw}) \\ q(\mathrm{TwA}) + q(\mathrm{TwC}) + q(\mathrm{TwG}) + q(\mathrm{TwT}) = l(\mathrm{Tw}) \\ q(\mathrm{AwA}) + q(\mathrm{CwA}) + q(\mathrm{GwA}) + q(\mathrm{TwA}) = r(\mathrm{wA}) \\ q(\mathrm{AwC}) + q(\mathrm{CwC}) + q(\mathrm{GwC}) + q(\mathrm{TwC}) = r(\mathrm{wC}) \\ q(\mathrm{AwG}) + q(\mathrm{CwG}) + q(\mathrm{GwG}) + q(\mathrm{TwG}) = r(\mathrm{wG}) \\ q(\mathrm{AwT}) + q(\mathrm{CwT}) + q(\mathrm{GwT}) + q(\mathrm{TwT}) = r(\mathrm{wT}) \end{cases} \tag{27.9}$$

From (27.9), one notices that the right-hand side cannot be set arbitrarily but must satisfy

$$\begin{aligned} l(\mathrm{Aw}) + l(\mathrm{Cw}) + l(\mathrm{Gw}) + l(\mathrm{Tw}) &= \sum_{L,R\in\{\mathrm{A,C,G,T}\}} q(\mathrm{LwR}) \\ &= r(\mathrm{wA}) + r(\mathrm{wC}) + r(\mathrm{wG}) + r(\mathrm{wT}) \end{aligned} \tag{27.10}$$

Inspection of (27.9) also indicated that for each w, a decoupled system is formed. In fact, for each $\mathrm{w}_i$, the unknowns $q(\mathrm{Lw}_i\mathrm{R})$ for different L and R only can occur in the constraints (27.9) for that particular $\mathrm{w}_i$, and never will occur in the constraints for $\mathrm{w}_j$, $j \neq i$. This is because $\mathrm{L}_i\mathrm{w}_i\mathrm{R}_i$ never can be equal to $\mathrm{L}_j\mathrm{w}_j\mathrm{R}_j$ for any possible $\mathrm{L}_i$, $\mathrm{L}_j$, $\mathrm{R}_i$, and $\mathrm{R}_j$. Obviously, the objective function in (27.8) already is decoupled for each $\mathrm{w}_i$, as each term in the objective function involves only one $q(\mathrm{Lw}_i\mathrm{R})$. Hence, we observe that the optimization problem (27.8) can be decoupled into $4^{k-2}$ subproblems of the form (27.9). ■

Problem (27.8) now can be solved readily. To solve the subproblems, let us rewrite them as follows:

$$\begin{aligned} &\text{maximize } -\sum_{i,j=1}^{4} p_{ij} \log p_{ij} \\ &\text{subject to } \begin{cases} p_{i1} + p_{i2} + p_{i3} + p_{i4} = l_i, & i = 1,2,3,4 \\ p_{1j} + p_{2j} + p_{3j} + p_{4j} = r_j, & j = 1,2,3,4 \\ p_{ij} \geq 0, & i,j = 1,2,3,4 \end{cases} \end{aligned} \tag{27.11}$$

where $p_{ij}$ are the unknowns $q(\mathrm{LwR})$ to be sought in (27.9).

**Theorem 27.1** [8] The solution to (27.11) is

$$p_{ij} = \begin{cases} \dfrac{l_i r_j}{\sigma}, & \text{if } \sigma \neq 0 \\ 0, & \text{if } \sigma = 0 \end{cases} \tag{27.12}$$

where $\sigma \equiv l_1 + l_2 + l_3 + l_4 = r_1 + r_2 + r_3 + r_4$ (*cf.* (27.10)).

***27.2.2.6 Denoising Formulas.*** In this section, we derive some new denoising formulas that maximize the entropy. Two approaches are introduced:

1. The first approach is to apply existing formulas such as (27.4) and (27.5) in the left-hand side of (27.9) to derive the right-hand-side functions $l(\cdot)$ and $r(\cdot)$, respectively.
2. The second approach is to apply existing formulas directly to the right-hand side of (27.9).

For the first approach, two formulas are obtained.

**Corollary 27.1** [8] For any 1-strings Y and Z and any $(k-2)$-string w,

$$q(\text{YwZ}) = \frac{f(\text{Yw})f(\text{wZ})}{f(\text{w})} \tag{27.13}$$

Formula (27.13) is identical to (27.4). Thus, we formally have proved the claim in [26] that formula (27.4) satisfies the maximum entropy principle.

Let us examine Yu *et al.*'s formula (27.5) to see whether a new denoising formula can be derived.

**Corollary 27.2** [8] For any 1-strings Y and Z and any $(k-2)$-string w,

$$q(\text{YwZ}) = \frac{1}{4\sigma}\left[f(\text{Yw}) + f(Y)\sum_{\text{R}} f(\text{wR})\right]\left[f(\text{wZ}) + f(Z)\sum_{\text{L}} f(\text{Lw})\right] \tag{27.14}$$

where

$$\sigma = \frac{1}{2}\left[\sum_{\text{L}} f(\text{Lw}) + \sum_{\text{R}} f(\text{wR})\right]$$

This formula, which satisfies the maximum entropy principle, is different from (27.5).

Our second approach to create new formulas stem from the following observation.

**Lemma 27.2** [8] *For all* $(k-1)$*-strings* w *and 1-strings* L *and* R*, let* $l(\mathrm{Lw}) = \alpha q(\mathrm{Lw})$ *and* $r(\mathrm{wR}) = \beta q(\mathrm{wR})$*, where* $\alpha$ *and* $\beta$ *are treated as normalization constants to fulfill the equality condition of* (27.10)*. Then by* (27.12),

$$q(\mathrm{LwR}) = \frac{q(\mathrm{Lw})q(\mathrm{wR})}{q(\mathrm{w})} \tag{27.15}$$

*To obtain two other new formulas, all we have to do is, to substitute formulas* (27.4) *and* (27.5) *into the right-hand side of* (27.15)*. One can check easily that* (27.15) *satisfies* (27.10).

**Corollary 27.3** [8] Let w be YxZ, where x is a $(k-4)$ string, and Y and Z are 1-strings. Then by using Hao *et al.*'s formula (27.4), (27.15) becomes

$$q(\mathrm{LYxZR}) = \begin{cases} \dfrac{f(\mathrm{LYx})f(\mathrm{YxZ})f(\mathrm{x})f(\mathrm{YxZ})f(\mathrm{xZR})}{[f(\mathrm{Yx})]^2[f(\mathrm{xZ})]^2} & \text{if } f(\mathrm{Yx})f(\mathrm{xZ}) \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{27.16}$$

**Corollary 27.4** [8] Let w be YxZ, where x is a $(k-4)$ string, and Y and Z are 1-strings. Then by using Yu *et al.*'s formula (27.5), (27.15) becomes

$$q(\mathrm{LYxZR}) = \begin{cases} \dfrac{[f(\mathrm{L})f(\mathrm{YxZ}) + f(\mathrm{LYx})f(\mathrm{Z})][f(\mathrm{Y})f(\mathrm{xZR}) + f(\mathrm{YxZ})f(\mathrm{R})]}{2[f(\mathrm{Y})f(\mathrm{xZ}) + f(\mathrm{Yx})f(\mathrm{Z})]} \\ \qquad\qquad \text{if } f(\mathrm{Y})f(\mathrm{xZ}) + f(\mathrm{Yx})f(\mathrm{Z}) \neq 0 \\ 0, \qquad \text{otherwise} \end{cases} \tag{27.17}$$

We remark that only nucleotide sequences were considered here. The decoupled constraint matrices of the optimization problem are thereby of size $(2 \times 4^{k-1})$ by $4^k$. If amino acid sequences are considered, then the decoupled systems will be of size $(2 \times 20^{k-1})$ by $20^k$. However, denoising formulas still can be derived similarly, and their forms will be the same.

As observed, different right-hand-side functions $l(\mathrm{v})$ and $r(\mathrm{v})$ in (27.9) provide different denoising formulas. In this work, we provide two approaches for defining them. For the four datasets tested in this work, formula (27.16) and formula (27.17) each have their own merits. We note that only Hao *et al.*'s formula (27.4) and Yu *et al.*'s formula (27.5) were used in constructing the right-hand sides. One may use other existing formulas to construct new denoising formulas via our two approaches.

### 27.2.3 Distance Measure

Let $n = 4^k$ be the length of the composition vector, and $\mathbb{S}$ be the set of composition vectors. To find the evolutionary distance between two species $\mathcal{A}$ and $\mathcal{B}$, we will compute the distance $d(\mathbf{a}, \mathbf{b})$ between their composition vectors $\mathbf{a} = (a_i)_{i=1}^n$ and $\mathbf{b} = (b_i)_{i=1}^n \in \mathbb{S}$, respectively. This distance then is used to represent the distance between their corresponding species. Assume the reciprocal of the length of the composition vector is computable, and assume there are no composition vectors $\mathbf{c}$ and $\mathbf{d}$ in $\mathbb{S}$ such that

$$\mathbf{c} = -\mathbf{d} \tag{27.18}$$

This assumption is reasonable, as (27.18) rarely will occur in real applications (*e.g.*, see (27.21)).

Once the conversion of sequences into frequency vectors was established, a variety of distances $d(\mathbf{a}, \mathbf{b})$ were calculated immediately. In the following, we will introduce some angle-based distance measures, which are used widely in practice [3, 25, 40, 47, 48, 52, 53]. We remark that those distances must satisfy the following conditions:

1. Nonnegativity $0 \le d(\mathbf{a}, \mathbf{b}) < +\infty$ for all $\mathbf{a}$ and $\mathbf{b} \in \mathbb{S}$
2. Identity of indiscernibles $d(\mathbf{a}, \mathbf{b}) = 0$ if and only if $\mathbf{a} = \mathbf{b}$
3. Symmetry $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$ for all $\mathbf{a}$ and $\mathbf{b} \in \mathbb{S}$

But the "triangle inequality" of the "metric distance"

$$d(\mathbf{a}, \mathbf{b}) \le d(\mathbf{a}, \mathbf{c}) + d(\mathbf{c}, \mathbf{b}), \qquad \forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{S}, \tag{27.19}$$

is not required for those distances.

#### *27.2.3.1 Angle-Based Distance.*

To measure the distance between the composition vectors $\mathbf{a}$ and $\mathbf{b} \in \mathbb{S}$, it is common to employ the cosine of their angle defined as follows [2]:

$$\cos\theta = \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\| \times \|\mathbf{b}\|} \tag{27.20}$$

where $\|\cdot\|$ is the Euclidean vector norm (*i.e.*, $\|\mathbf{a}\| = \sqrt{\sum_{i=1}^n (a_i)^2}$ ).

Stuart *et al.* [47, 48] were the first to introduce the angle distance for the phylogenetic analysis. A formula is given by

$$d^{\text{Stuart}}(\mathbf{a}, \mathbf{b}) = -\log\left(\frac{1+\cos\theta}{2}\right) = -\log\left[\frac{1}{2}\left(1 + \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}\right)\right] \tag{27.21}$$

Formula (27.21) is a distance on the set $\mathbb{S}$. Using the Cauchy–Schwarz inequality, one can show that $d^{\text{Stuart}}$ satisfies the first and the second conditions of a distance. Moreover, $d^{\text{Stuart}}$ satisfies the third condition. Because of characteristics of the logarithm function, $d^{\text{Stuart}}(\mathbf{a}, \mathbf{b})$ can be sufficiently large if the angle between **a** and **b** is large enough. For this reason, this distance can be used for the phylogenetic analysis of the dataset in which the species are far away from each other. In fact, formula (27.21) has been applied successfully for the phylogenetic analysis of whole genomes of bacteria and vertebrates [45, 46, 47, 48, 55].

Hao *et al.* [25, 40] proposed the following formula:

$$d^{\text{Hao}}(\mathbf{a}, \mathbf{b}) = \frac{1 - \cos\theta}{2} = \frac{1}{2}\left(1 - \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\| \times \|\mathbf{b}\|}\right) \tag{27.22}$$

One can verify that this measure is a distance satisfying the three conditions. Because the cosine value computed by (27.20) varies between $-1$ and 1, the function value $d^{\text{Hao}}(\mathbf{a}, \mathbf{b})$ is normalized to the interval (0, 1). Formula (27.22) is used widely and has achieved great success in the phylogenetic analysis of whole genomes of bacteria, viruses, and vertebrates [11, 12, 20, 21, 29, 40, 58].

Although Hao *et al.*'s distance is defined based on the cosine of the angle, it has a close relationship with the Euclidean distance. To see how it works, let us take two vectors **c** and $\mathbf{d} \in \mathbb{R}^2$. The angle $\theta$ is a one-to-one mapping of the following vector:

$$\frac{\mathbf{c}}{\|\mathbf{c}\|} - \frac{\mathbf{d}}{\|\mathbf{d}\|}$$

Moreover, for the length of this vector, we have, by the law of cosines, [6] that

$$\left\| \frac{\mathbf{c}}{\|\mathbf{c}\|} - \frac{\mathbf{d}}{\|\mathbf{d}\|} \right\|^2 = 2 - 2\cos\theta$$

Generally, we have the following property for Hao *et al.*'s distance (27.22). Given any vectors **a** and $\mathbf{b} \in \mathbb{S}$, their Hao *et al.*'s distance relates to the Euclidean distance between their normalized vectors $\frac{\mathbf{a}}{\|\mathbf{a}\|}$ and $\frac{\mathbf{b}}{\|\mathbf{b}\|}$ as follows:

$$\mathrm{d}^{\text{Hao}}(\mathbf{a}, \mathbf{b}) = \frac{1}{4}\left\| \frac{\mathbf{a}}{\|\mathbf{a}\|} - \frac{\mathbf{b}}{\|\mathbf{b}\|} \right\|^2 \tag{27.23}$$

It can be observed from (27.23) that Hao *et al.*'s distance is the square of a Euclidean distance and thereby does not satisfy the triangle inequality. If the triangle inequality further is required for the distance, then we can define

$$d^{\text{NUD}}(\mathbf{a}, \mathbf{b}) = \frac{1}{2}\left\| \frac{\mathbf{a}}{\|\mathbf{a}\|} - \frac{\mathbf{b}}{\|\mathbf{b}\|} \right\| \tag{27.24}$$

(*i.e.*, the Euclidean distance between their normalized vectors). This distance satisfies all conditions of a "metric distance," and is the square root of Hao *et al.*'s distance. In addition, we directly can use the angle to measure the distance. Define

$$d^{\text{angle}}(\mathbf{a}, \mathbf{b}) = \frac{1}{\pi} \arccos \left( \frac{\mathbf{a}^T \mathbf{b}}{\|\mathbf{a}\| \times \|\mathbf{b}\|} \right) \tag{27.25}$$

We see

$$d^{\text{angle}}(\mathbf{a}, \mathbf{b}) \in (0, \quad 1)$$

for all vectors $\mathbf{a}, \mathbf{b} \in \mathbb{S}$, and $d^{\text{angle}}(\mathbf{a}, \mathbf{b})$ is a "metric distance." Newly defined distances (27.24) and (27.25) will be tested on more realistic datasets (*e.g.*, see Section 27.3.4).

### 27.2.4 Phylogenetic Tree Construction

Given the molecular sequences for any $n$ species $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n$, $n \geq 4$, we construct their frequency vectors, and then the composition vectors $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_n$. The distance $d_{ij}$, $i, j = 1, 2, \cdots, n$, between any two composition vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ can be obtained by the angle-based distance measure described in Section 27.2.3. A distance matrix consists of the collection of the pairwise distances for all $n$ species and is given by

$$\begin{array}{c c}
 & \begin{array}{ccccc} \mathcal{C}_1 & \mathcal{C}_2 & \mathcal{C}_3 & \cdots & \mathcal{C}_n \end{array} \\
\begin{array}{c} \mathcal{C}_1 \\ \mathcal{C}_2 \\ \mathcal{C}_3 \\ \vdots \\ \mathcal{C}_n \end{array} &
\begin{bmatrix}
0 & d_{12} & d_{13} & \cdots & d_{1n} \\
d_{21} & 0 & d_{23} & \cdots & d_{2n} \\
d_{31} & d_{32} & 0 & \cdots & d_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
d_{n1} & d_{n2} & d_{n3} & \cdots & 0
\end{bmatrix}
\end{array}$$

Any distance-based phylogenetic tree construction method may be employed to build the tree, for instance, the Fitch–Margoliash (FM) method [19], the unweighted pair group method with arithmetic mean method (UPGMA) [51], the neighbor-joining (NJ) method [41], and so on. It is worth mentioning that the FM method is not feasible when the number of species is larger than 100, and the information about the branch length of the tree is not available if the UPGMA method is used. In this work, all trees will be drawn by the NJ method. This algorithm is available in several software packages, for instance, PHYLIP [18], SPLITSTREE [27], MEGA [49], and so on.

## 27.3 APPLICATION AND DICUSSION

Four worked examples are given. Until otherwise stated, for the purpose of distance measurement, Equation (27.23) is used throughout all computational experiments. All figures are produced by MEGA and SPLITSTREE.

### 27.3.1 Example 1

The phylogenetic relationship among tetrapods has been discussed widely in the area of phylogeny and evolution. One early topic was whether birds are related more closely to crocodilians or to mammals. Based on the traditional classification and the results that stemmed from a large amount of molecular, morphological, and paleontological data, birds are thought to be grouped with crocodilians. However, many studies based on 18S rRNA sequences supported the grouping of birds and mammals [56]. Using the CV method without denoising and with each of the five denoising formulas, every taxa were grouped to their corresponding amphibian, reptile, bird, or mammal clade. However, inspection of Figure 27.4 indicated that with the denoising formula (27.16), the bird and reptile clades were grouped together, and the two oryctolaguses of the mammal clade are well grouped. When the CV method was used without denoising, or with the denoising formulas (27.4), (27.5), (27.14), or (27.17), birds were grouped with mammals, and the two oryctolaguses were not grouped together. For further discussion, see [9].

### 27.3.2 Example 2

The characteristics of HBV genotype C subgroups in Hong Kong and their relationship with HBV genotype C in other parts of Asia were investigated by Chan *et al.* [7]. The full genome nucleotide sequences of 49 HBV genotype C isolated from Hong Kong local patients, together with 69 published HBV genotype C and 12 well-known HBV nongenotype C first were collected.

The multiple sequence alignment method [50] was used to align those sequences and the distance matrix then was obtained. One phylogenetic tree, called the “NJ tree,” thereby was constructed by the NJ method [41, 49]. In their NJ tree, the HBV genotype C were divided into two subgroups:

- The genotype Ce
- The genotype Cs

Using the CV method without denoising, or with any denoising formula (27.4), (27.5), (27.14), or (27.17), respectively, every HBV of a different genotype was grouped correctly to its corresponding genotype subgroup. In particular, the 49 genotype C isolated from Hong Kong were separated identically into two subgroups, genotype Ce and genotype Cs, where 39 isolates ($\sim$ 80%) belonged to the genotype Cs subgroup, and 10 isolates ($\sim$ 20%) belonged to the genotype Ce subgroup. Using

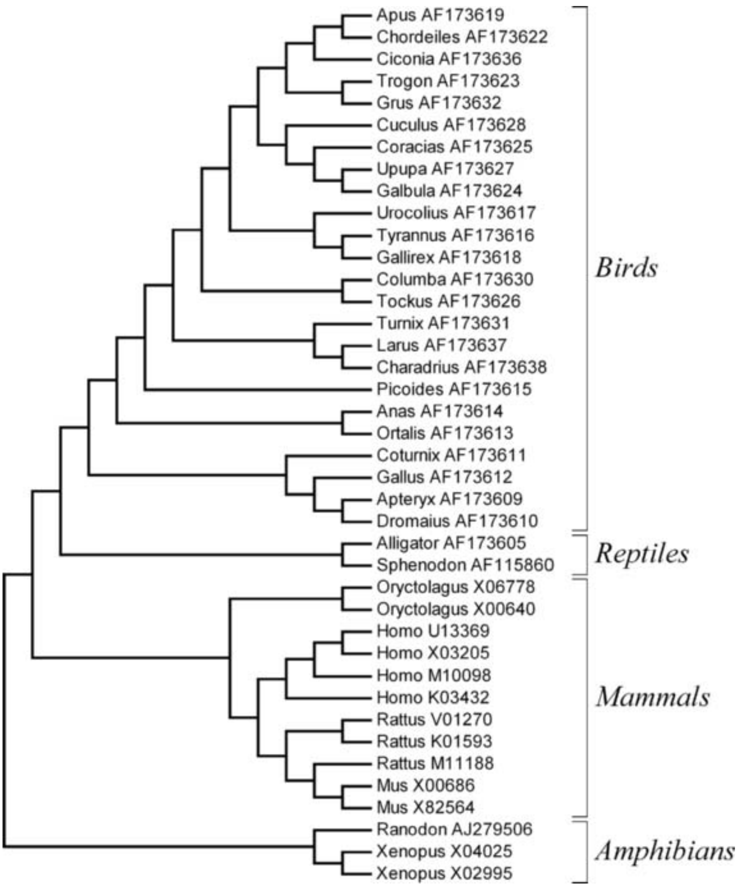


**Figure 27.4** The CV tree ($k = 10$) based on the 18S rRNA dataset of tetrapods analyzed by Xia *et al.* [56].

(27.13), (27.14), and (27.17), the phylogenetic tree is the same. The CV tree with denoising formulas (27.14) was shown in Figure 27.5.

### 27.3.3 Example 3

In our third set of computational experiments, we have applied two denoising formulas (27.14) and (27.17) to the set of 20 complete mtDNA sequences, including six primates, eight ferungulates (artiodactyls + cetaceans + perissodactyls + carnivores), two rodents, and three outgroups (marsupials and monotremes), which is the same set of species as, for exampley, [5].

The phylogenetic relationship among mammals has been a long-standing problem in the area of phylogeny and evolution. Using (27.14) and (27.17), the phylogenetic tree is the same. Figure 27.6 shows the phylogenetic trees calculated by (27.17) with

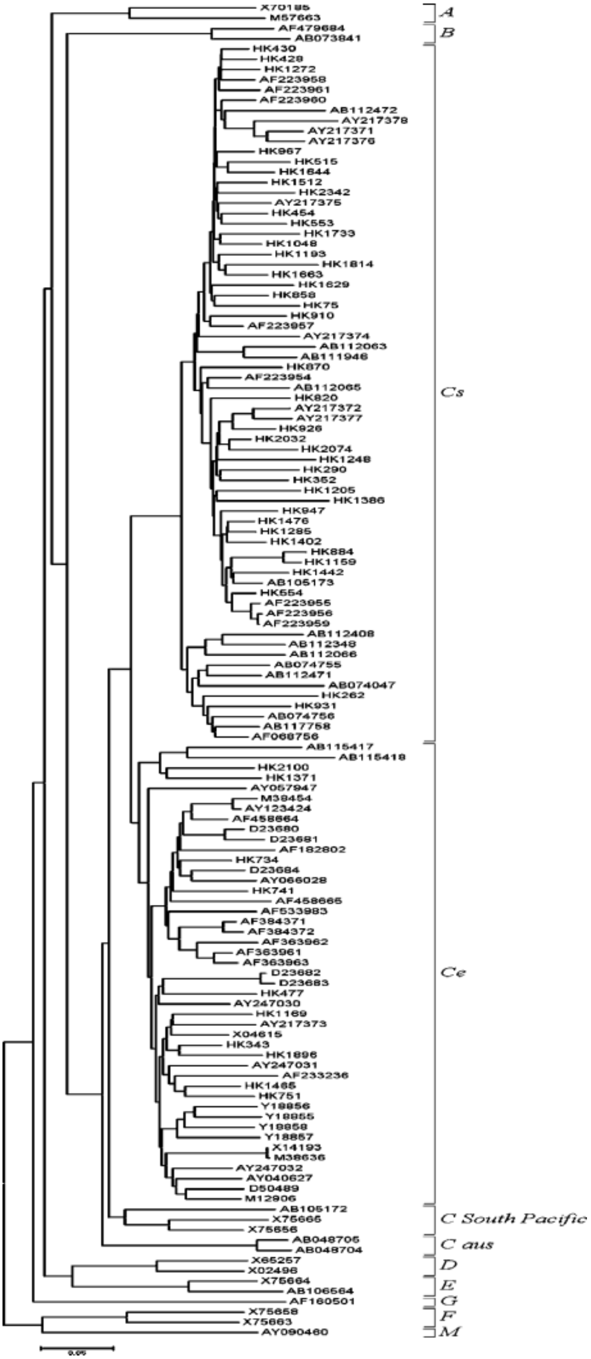


**Figure 27.5** The CV tree ($k = 11$) based on the dataset of complete nucleotide sequences of HBV analyzed by Chan *et al.* [7].

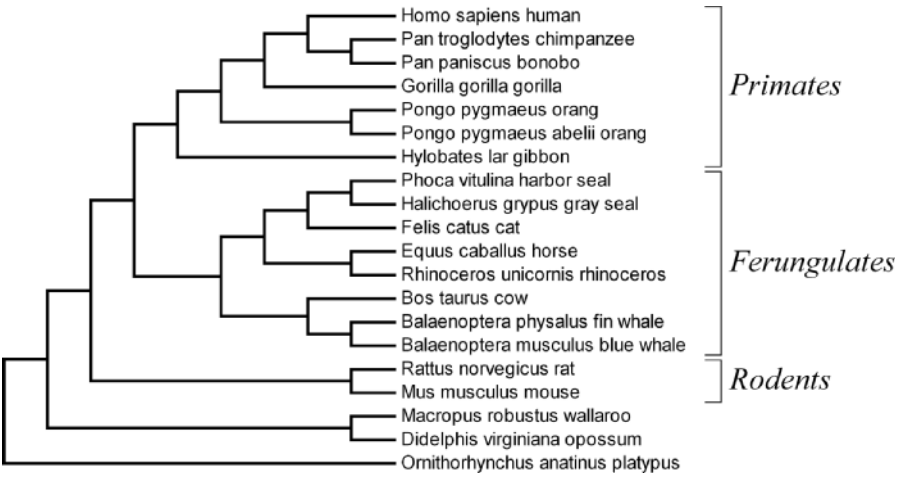

Homo sapiens human
Pan troglodytes chimpanzee
Pan paniscus bonobo
Gorilla gorilla gorilla
Pongo pygmaeus orang
Pongo pygmaeus abelii orang
Hylobates lar gibbon
Phoca vitulina harbor seal
Halichoerus grypus gray seal
Felis catus cat
Equus caballus horse
Rhinoceros unicornis rhinoceros
Bos taurus cow
Balaenoptera physalus fin whale
Balaenoptera musculus blue whale
Rattus norvegicus rat
Mus musculus mouse
Macropus robustus wallaroo
Didelphis virginiana opossum
Ornithorhynchus anatinus platypus
Primates
Ferungulates
Rodents

**Figure 27.6** The CV tree ($k = 14$) based on the dataset of complete mtDNA sequences of a mammal analyzed by Cao *et al.* [5].

$k = 14$. The result of (27.17) is identical to the one done in [5]. In Figure 27.6, four trees among primates, ferungulates, rodents, and the outgroup are well grouped using the CV method.

### 27.3.4 Example 4

In our last computational experiments, we applied a denoising formula (27.17) to the set of 34 choroplast genomes (or complete protein genome sequences), including two archaea, seven chlorophyte *s.l.*, eight eubacteria, three eukaryote, one glaucophyte, four rhodophyte *s.l.*, and nine seed plants, which is the same set of species as [12, 58].

Figure 27.7 shows the phylogenetic trees calculated by (27.17) with $k = 6$. The result of (27.17) mostly agrees with the one in [58]. Some salient features of Figure 27.7 can be summarized as follows:

- Based on the widely accepted endosymbiotic theory that chloroplasts sprang from a cyanobacteria-like ancestor [22, 23, 33], all the chloroplast genomes yield a clade branched in the domain of eubacteria and are diverged from a most recent common ancestor at cyanobacteria. Our denoising formula can identify cyanobacteria as the most closely related prokaryotes of chloroplasts, even though massive gene transfering from the endosymbiont to the nucleus of the host cell was found [30, 31, 32].
- The chloroplasts are divided into two major clades:
  1. The green plants sensu lato, or chlorophytes *s.l.* [38] include all taxa with a chlorophyte chloroplast, both primary and secondary endosymbioses in origin.

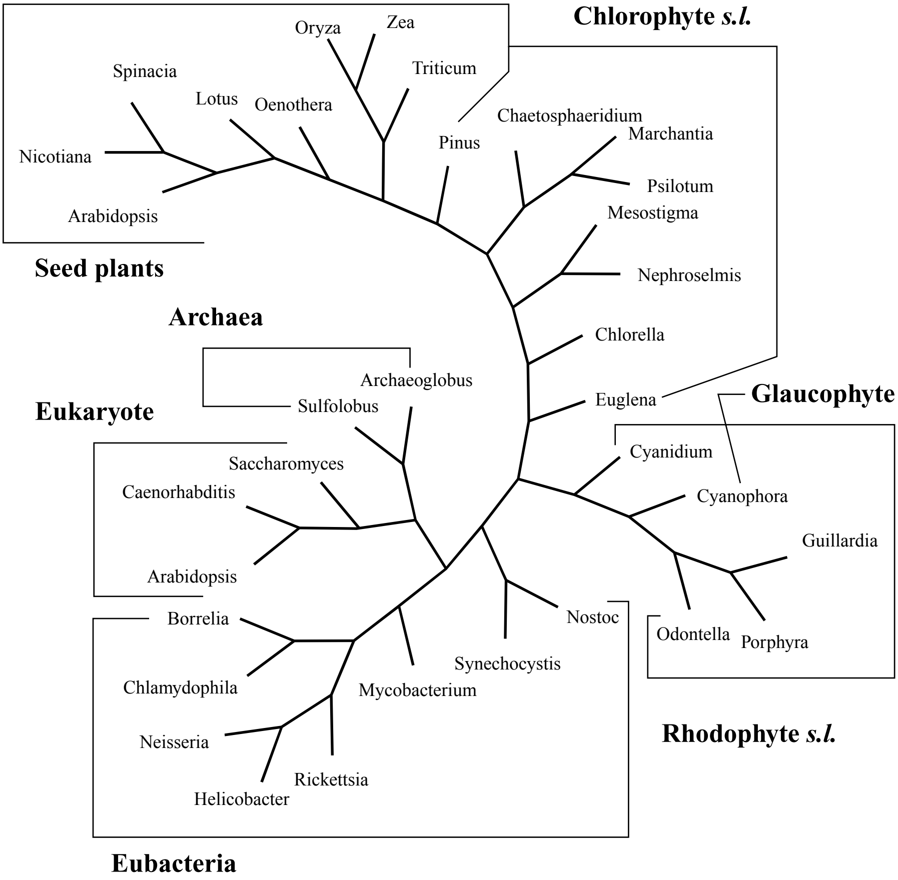


**Figure 27.7** The CV tree ($k = 6$) based on the dataset of complete protein genome sequences of a choroplast analyzed by Yu *et al.* [58].

2. The glaucophyte Cyanophora and members of rhodophytes *s.l.* refer to rhodophytes (or red algae, Cyanidium, and Porphyra in the tree) and their secondary symbiotic derivatives (the heterokont Odontella and the cryotphyte Guillarida).

- Inspection of Figure 27.7 shows that Cyanophora is mixed into rhodophytes *s.l.*. These findings have been reported in [13, 44], despite the fact that the glaucophyte (Cyanophora is grouped into glaucophyte) represents the earliest branch in chloroplast evolution with the green plants *s.l.* and rhodophytes *s.l.* as sister taxa [1, 31, 32, 35]
- In chlorophyte *s.l.*, the green algae (*i.e.*, chlorella, Mesostigma, and Nephroselmis) and euglena are basal to that lineage, and the seed plants cluster together as a derived group. But, the relationships among the other taxa (*i.e.*, Marchantia, Psilotum, and Chaetosphaeridium) deviate slightly from our

traditional understanding, probably because of limited taxon sampling in these primitive green plants [58].

- Similar to the result of [12], chlorella is connected between euglena and a clade of Mesostigma and Nephroselmis.

As a check, different distance formulas (27.21), (27.23), (27.24), and (27.25) were used, and the results of each formula showed no sizable differences. In addition, they yielded the same phylogenetic tree.

## 27.4 CONCLUDING REMARKS

In this chapter, one kind of alignment-free method, namely, the CV method, was introduced. Compared with the multiple sequence alignment methods that are employed widely, the CV method has several advantages. For instance, it can be used for phylogenetic analysis of whole genome sequences of bacteria, viruses, and so on [12, 40, 58], in which the sequence alignment methods all failed. Our denoising formulas worked well in the classification of HBV, mammal, and chloroplast. As a systematic method for studying the classification of species, no scoring matrix or gap penalty [54] was required by the CV method. For computing the distance between two species, its operation cost is $\mathcal{O}\big(N \log N\big)$, and the memory requirement is $\mathcal{O}(N)$, where $N$ is the length of the longer sequence. With the development of sequence techniques, more and more complete genome sequences are available, and these advantages are becoming more important or even necessary for sequence comparison methods.

The method described in this work has been written in MATLAB for the preparation of input data and in FORTRAN 90 for the rest of the numerical computations. The program is distributed by R.W. Wang and J.C.-F. Wong and can be obtained by an anonymous ftp from our ftp website at the following website: http://www.math.cuhk.edu/~jwong. For example, the current version of the program allows a maximum of 34 library species to be analyzed on 64-byte machines (compilers).

## REFERENCES

1. J. Adachi, P.J. Waddell, W. Martin, and M. Hasegawa. Plastid genome phylogeny and a model of amino acid substitution for proteins encoded by chloroplast DNA, *J Mol Evol*, 50:348–358, 2000.
2. M.W. Berry, Z. Drmac, and E.R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Rev*, 41:335–362, 1999.
3. B.E. Blaisdell. A measure of the similarity of sets of sequences not requiring sequence alignment. *Proc Natl Acad Sci*, 83:5155–5159, 1986.
4. V. Brendel, J.S. Beckmann, and E.N. Trifonov. Linguistics of nucleotide sequences: Morphology and comparison of vocabularies. *J Biomol Struct Dyn*, 4:11–20, 1986.

5. Y. Cao, A. Janke, P.J. Waddell, M. Westerman, O. Takenaka, S. Murata, N. Okada, S. Paabo, M. Hasegawa. Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. *J Mol Evol*, 47:307–322, 1998.
6. J. Casey. A Treatise on Spherical Trigonometry: And Its Application to Geodesy and Astronomy with Numerous Examples. Longmans, Green, and Company, London, UK, 1889.
7. H.L.Y. Chan, C.H. Tse, E.Y.T. Ng, K.S. Leong, K.H. Lee, S.K.W. Sui, and J.J.Y. Song. Epidemiological and virological characteristics of 2 subgroups of Hepatitis B virus genotype C. *J Infect Dis*, 191:2022–2032, 2005.
8. R.H. Chan, T.H. Chan, and R.W. Wang. Composition vector method based on maximum entropy principle for sequence comparison. Research ReportMATH-09-05 (367), Department of Mathematics, The Chinese University of Hong Kong.
9. R.H. Chan, T.H. Chan, and R.W. Wang. Composition vector method based on maximum entropy principle for sequence comparison. To appear.
10. R.L. Charlebois, R.G. Beiko, and M.A. Ragan. Microbial phylogenomics: Branching out. *Nature*, 421:217, 2003.
11. K.H. Chu, C.P. Li, and J. Qi. Ribosomal RNA as molecular barcodes: A simple correlation analysis without sequence alignment. *Bioinformatics*, 22:1690–1710, 2006.
12. K.H. Chu, J. Qi, Z.G. Yu, and V. Anh. Origin and phylogeny of chloroplasts: A simple correlation analysis of complete genomes. *Mol Biol Evol*, 21:200–206, 2004.
13. J. De Las Rivas, J.J. Lozano, and A.R. Ortiz. Comparative analysis of chloroplast genomes: Functional annotation, genome-based phylogeny, and deduced evolutionary patterns. *Genome Res*, 12:567–583, 2002.
14. W.F. Doolittle. Phylogenetic classification and the universal tree. *Science*, 284:2124–2128, 1999.
15. R.C. Edgar. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, 32(5):1792–1797, 2004.
16. J.A. Eisen and C.M. Fraser. Phylogenomics: Intersection of evolution and genomics. *Science*, 300:1706–1707, 2003.
17. D.F. Feng and R.F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol*, 25:351–360, 1987.
18. J. Felsenstein. *PHYLIP (phylogeny inference package) version 3.5c.* http://evolution.genetics.washington.edu/phylip.html.
19. W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.
20. L. Gao and J. Qi. Whole genome molecular phylogeny of large dsDNA viruses using composition vector method. *BMC Evol Biol*, 7:1–7, 2007.
21. L. Gao, J. Qi, H. Wei, Y. Sun, and B.L. Hao. Molecular phylogeny of coronaviruses including human SARS-CoV. *Chin Sci Bull*, 48:1170–1174, 2003.
22. M.W. Gray. The endosymbiont hypothesis revisited. *Int Rev Cytol*, 141:233–357, 1992.
23. M.W. Gray. Evolution of organellar genomes. *Curr Opin Genet Dev*, 9:678–687, 1999.
24. S. Grumbach and F. Tahi. Compression of DNA sequences. *Data Compression Conference*. IEEE Computer Society Press, Sbowbird, Utah.
25. B.L. Hao, J. Qi, and B. Wang. Prokaryotic phylogeny based on complete genomes without sequence alignment. *Mod Phys Lett B*, 2:1–4, 2003.

26. R. Hu and B. Wang. Statistically significant strings are related to regulatory elements in the promoter regions of Saccharomyces cerevisiae. *Physica A*, 290:464–474, 2001.

27. D.H. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Mol Biol Evol*, 23(2):254–267, 2005.

28. M. Li, J.H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang. An information-based sequence distance and its application to hole mitochondrial genome phylogeny. *Bioinformatics*, 17:149–154, 2001.

29. G. Lu, S. Zhang, and X. Fang. An improved string composition method for sequence comparison. *BMC Bioinformatics*, 9(Suppl 6):S15, 2008.

30. W. Martin and R.G. Herrmann. Gene transfer from organelles to the nucleus: How much, what happens, and why? *Plant Physiol*, 118:9–17, 1998.

31. W. Martin, B. Stoebe, V. Goremykin, S. Hansmann, M. Hasegawa, and K.V. Kowallik. Gene transfer to the nucleus and the evolution of chloroplasts. *Nature*, 393:162–165, 1998.

32. W. Martin, T. Rujan, E. Richly, A. Hansen, S. Cornelsen, T. Lins, D. Leister, B. Stoebe, M. Hasegawa, and D. Penny. Evolutionary analysis of Arabidopsis, cyanobacterial, and chloroplast genomes reveals plastid phylogeny and thousands of cyanobacterial genes in the nucleus. *Proc Natl Acad Sci U S A*, 99:12246–12251, 2002.

33. G.I. McFadden. Chloroplast origin and integration. *Plant Physiol*, 125:50–53, 2001b.

34. S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*, Springer-Verlag, London, UK, 1993.

35. D. Moreira, H. LE Guyader, and H. Ppilippe. The origin of red algae and the evolution of chloroplasts. *Nature*, 405:69–72, 2000.

36. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48:443–453, 1970.

37. C. Notredame, D.G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–217, 2000.

38. J.D. Palmer and C.F. Delwiche. The origin and evolution of plastids and their genomes. In D.E. Soltis, P.S. Soltis, and J.J. Doyle, editors, *Molecular Systematics of Plants II DNA Sequencing*. Kluwer, London, UK, 1998, pp. 345–409.

39. P.A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, Cambridge, MA, 2000, p. 75.

40. J. Qi, B. Wang, and B.L. Hao. Whole proteome prokaryote phylogeny without sequence alignment: A k-string composition approach. *J Mol Evol*, 58(1):1–11, 2004.

41. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4(4):406–425, 1987.

42. T.F. Smith and M.S. Waterman. Identification of common molecular sequences. *J Mol Biol*, 147:195–197, 1981.

43. B. Snel, P. Bork, and M.A. Huynen. Genome phylogeny based on gene content. *Nat Genet*, 21:108–110, 1999.

44. V.L. Stirewalt, C.B. Michalowski, W. Loffelhardt, H.J. Bohnert, and D.A. Bryant. Nucleotide sequence of the cyanelle genome from Cyanophora paradoxa. *Plant Mol Biol Rep*, 13:327–332, 1995.

45. G.W. Stuart and M.W. Berry. A comprehensive whole genome bacterial phylogeny using correlated peptide motifs defined in a igh dimensional vector space. *J Bioinform Comput Biol*, 1(3):475–493, 2003.

46. G.W. Stuart and M.W. Berry. An SVD-based comparison of nine whole eukaryotic genomes supports a coelomate rather than ecdysozoan lineage. *BMC Bioinformatics*, 5:204, 2004.

47. G.W. Stuart, K. Moffett, and S. Baker. Integrated gene and species phylogenies from unaligned whole genome protein sequences. *Bioinformatics*, 62:100–108, 2002.

48. G.W. Stuart, K. Moffett, and J.J. Leader. A comprehensive vertebrate phylogeny using vector representations of protein sequences from whole genomes. *Mol Biol Evol*, 19:554–562, 2002.

49. K. Tamura, J. Dudley, M. Nei, and S. Kumar. MEGA4: Molecular evolutionary genetics analysis (MEGA) software version 4.0. *Mol Biol Evol*, 24:1596–1599, 2007.

50. J.D. Thompson, D.G. Higgins, and T.J. Gibson. Clustal W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680, 1994.

51. UPGMA Software. http://pubmlst.org/software/analysis/start/manual/upgma.shtml.

52. S. Vinga and J. Almeida. Alignment free sequence comparison-a review. *Bioinformatics*, 19:513–523, 2003.

53. J. Wang and X. Zheng. WSE, a new sequence distance measure based on word frequencies. *Math Biosci*, 215:78–83, 2008.

54. M.S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman and Hall, New York, 1995.

55. X. Wu, X.F. Wan, G. Wu, D. Xu, and G. Lin. Phylogenetic analysis using complete signature information of whole genomes and clustered Neighbor-Joining method. *J Res Appl*, 2:219–248, 2006.

56. X. Xia, Z. Xie, and K.M. Kjer, 18S ribosomal RNA and tetrapod phylogeny. *Syst Biol*, 52:283–295, 2003.

57. H.M. Xie. *Grammatical Complexity and One-dimensional Dynamical Systems*. World Scientific, Singapore, 1996.

58. Z.G. Yu, L.Q. Zhou, V. Anh, K.H. Chu, S.C. Long, and J.Q. Deng. Phylogeny of prokaryotes and chloroplasts revealed by a simple composition approach on all protein sequences from whole genome without sequence alignment. *J Mol Evol*, 60:538–545, 2005.

# V

# MICROARRAY DATA ANALYSIS

# 28

# MICROARRAY GENE EXPRESSION DATA ANALYSIS

Alan Wee-Chung Liew and Xiangchao Gan

## 28.1 INTRODUCTION

Research in life science, in particular, the study of the genetic codes in living organisms, has reached its epic with the sequencing of the full human genome. With the availability of the vast number of genetic sequences, the next major challenge is to study gene activity or gene function. Important insights into gene function can be gained by gene expression profiling. Gene expressing profiling is the process of determining when and where particular genes are expressed. For example, some genes are turned on (expressed) or turned off (repressed) when there is a change in external conditions or stimuli. In multicellular organisms, gene expressions in different cell types is different during different developmental stages in life. Even within the same cell type, gene expressions are dependent on the cell cycle the cells are in. DNA mutation may alter the expression of certain genes, which causes illness such as abnormal tumor growth or cancer. Furthermore, the expression of one gene often is regulated by the expression of another gene. A detailed analysis of all this information will provide an understanding about the networking of different genes and their functional roles.

In the past, genes and their expression profiles are studied one at a time. However, this method is inadequate for the holistic study of the complete genome of an organism because the expressions of different genes are generally interdependent.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Microarray technology, which allows massively parallel, high-throughput profiling of gene expression in a single hybridization experiment, has emerged as a powerful tool for genomic research [19, 26]. By using an array containing many DNA samples, scientists can determine, in a single experiment, the expression levels of tens of thousands of genes within a cell by measuring the amount of mRNA bound to each site on the array. Besides the enormous scientific potential of DNA microarrays in the study of gene expressions, gene regulations, and interactions, they also have very important applications in pharmaceutical and clinical research. For example, by comparing the ways in which genes are expressed in a normal and in a diseased organ, scientists might be able to identify the genes and hence the associated proteins that are part of the disease process. Researchers then could use that information to synthesize drugs that interact with these proteins, thus reducing the disease's effect on the body.

## 28.2 DNA MICROARRAY TECHNOLOGY AND EXPERIMENT

DNA Microarrays are small, solid supports onto which sequences from tens of thousands of different genes are immobilized at fixed locations. The supports usually are made of glass microscope slides but also can be silicon chips or nylon membranes. The DNA is spotted or synthesized directly onto the support.

In spotted microarrays, the probes are oligonucleotides, cDNA, or small fragments of polymerase chain reaction (PCR) products that correspond to mRNAs. The probes are synthesized prior to the deposition on the array surface and then are "spotted" onto the glass using an array of fine pins or needles controlled by a robotic arm. In this chapter, we will concentrate our discussion on the spotted microarray.

In oligonucleotide microarrays, the probes are short sequences designed to match parts of the sequence of known or predicted open reading frames. Oligonucleotide arrays are produced by synthesizing short oligonucleotide sequences directly onto the array surface instead of depositing intact sequences. Sequences may be longer (60-*mer* probes such as the Agilent [Santa Clara, CA] design) or shorter (25-*mer* probes produced by Affymetrix [Santa Clara, CA]) depending on the desired purpose. One technique used to produce oligonucleotide arrays is photolithographic synthesis (Agilent and Affymetrix) on a silica substrate in which light and light-sensitive masking agents are used to build a sequence one nucleotide at a time across the entire array [24]. Each applicable probe is unmasked selectively prior to bathing the array in a solution of a single nucleotide, then a masking reaction takes place, and the next set of probes is unmasked in preparation for a different nucleotide exposure. This process is repeated until the probes are constructed fully.

In a two-channel microarray experiment, two samples of cRNA, which are reversed transcribed from mRNA purified from cellular contents, are labeled with different fluorescent dyes (usually cyanine3 (Cy3) and cyanine5 (Cy5)) to constitute the cDNA targets. The two cDNA targets then are hybridized onto a cDNA microarray. If a target contains a cDNA whose sequence is complementary to the DNA probe on a given spot, then that cDNA will hybridize to the spot where it will be

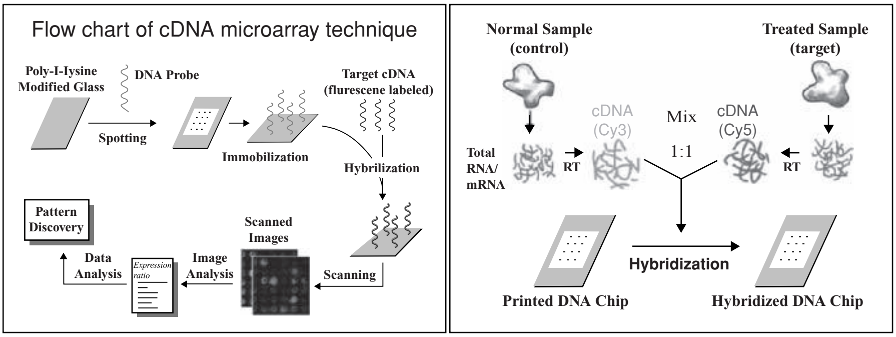


**Figure 28.1** Left: A schematic of the cDNA microarray technique. Right: The steps involved in a cDNA microarray experiment.

detectable by its fluorescence. Spots with more bound targets will fluoresce more intensely.

Once the cDNA targets have been hybridized to the array and any loose targets have been washed off, the array is laser scanned to determine how much of each targets is bound to each spot. The hybridized microarray is scanned for the red wavelength (at approximately 635 nm for Cy5) and the green wavelength (at approximately 530 nm for Cy3), which produces two sets of images typically in 16-bits iff format. The ratio of the two fluorescence intensities at each spot indicates the relative abundance of the corresponding DNA sequence in the two cDNA samples that are hybridized to the DNA probe on the spot. By examining the expression ratio of each spots in the Cy3 and Cy5 images, gene expression study can be performed. Figure 28.1 shows a schematic of the cDNA microarray technique (left) and the steps in performing a cDNA microarray experiment (right).

## 28.3 IMAGE ANALYSIS AND EXPRESSION DATA EXTRACTION

The goal in image analysis is to quantify automatically each spot giving information about the relative extent of hybridization of the two cDNA samples. However, this is a nontrivial task because of the poor contrast between spots and background, and the many contaminations or artifacts resulting from the hybridization procedures such as irregular spot shape and size, dust on the slide, large intensity variation within spots and background, and nonspecific hybridization. Accurate extraction of the gene expression data from the microarray image is important because it will impact the quality of the downstream data analysis.

A microarray image typically will contain $N \times M$ blocks, where each block will contain $p \times q$ spots. The $N \times M$ blocks on each array are printed simultaneously by repeatedly spotting the slide surface with $N \times M$ print tips. The relative placement of adjacent blocks therefore is determined by the spacing between adjacent print tips. Adjacent spots inside each block are separated during printing by slightly offsetting the print tips after each spotting. These spots must be segmented individually from the background to compute the expression ratio. Microarray image analysis generally involves several steps: (i) block segmentation in which each block within a microarray image is delineated, (ii) gridding in which the location of each spot within a block is determined, and (iii) spot extraction in which each spot is segmented and its intensity value is determined. Subsequently, we briefly outline the main steps involved in microarray image analysis in our algorithm called GENEICON [16].

### 28.3.1 Image Preprocessing

The input microarray images consist of a pair of 16-bit images in tiff format. For computational efficiency, a spot segmentation algorithm usually operates on a single image. Let $X$ denote the composite image obtained from $R$ (Cy5) and $G$ (Cy3), then $X$ can be computed as follows:

$$X = \lfloor 0.5 \times (G' + (\text{median}(G')/\text{median}(R'))R') \rfloor \tag{28.1}$$

where $G' = \sqrt{G}$, $R' = \sqrt{R}$, and $\lfloor \; \rfloor$ denotes rounding to the nearest integer in the range [0–255].

### 28.3.2 Block Segmentation

As the blocks in a microarray image are arranged in a rigid pattern, and each of the blocks in a microarray image is surrounded by regions void of any spots, an effective method for block segmentation is through an analysis of the vertical and horizontal image projection profiles. In GENEICON, the projection profiles are obtained from an adaptively binarized image. By performing analysis on the projection profiles, accurate block segmentation can be achieved (see Figure 28.2).

### 28.3.3 Automatic Gridding

The gridding strategy in GENEICON consists of first locating the good quality spots (called guide spots) and then inferring the geometry of the grid from these spots. To account for the variable background and spot intensities, a novel adaptive thresholding technique based on mathematical morphology is used to detect the guide spots. After the guide spots are found, global rotation is compensated for, and the correct grid parameters are estimated based on the spatial arrangement of the guide spots. Figure 28.3 shows an example of automatic gridding of a block in a microarray image.

### 28.3.4 Spot Extraction

Spot segmentation involves finding a circle that separates the spot from the background. The spot segmentation task consists of three steps: (i) background

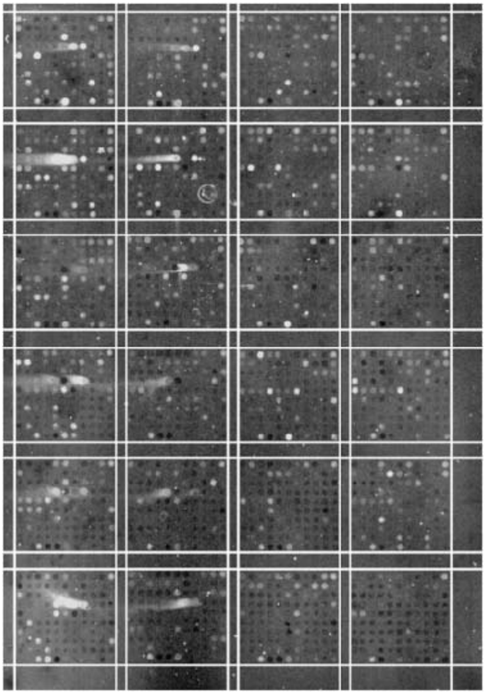


**Figure 28.2** The segmentation of a microarray image into blocks.

equalization for intensity variation in the subregion defined by the gridding, (ii) statistical intensity modeling and optimum thresholding of the subregion, and finally, (iii) finding the best-fit circle that segments the spot. A spot is assumed present if the ratio of the median intensity between the tentative spot pixels and the background pixels is larger than a preset value or when a guide spot is present within the subregion. When a spot is present, the intensity distribution of the pixels within the subregion is modeled using a two-class Gaussian Mixture Model to find the optimum

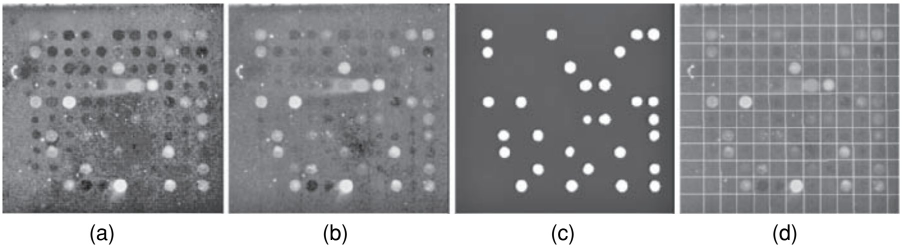


**Figure 28.3** (a): A block of spots from a microarray image shown as RGB color image, where the green component is given by Cy3, the red component is given by Cy5, and the blue component is set to zero. (b): The corresponding composite image. (c): The guide spots found. (d): Grid generated from the guide spot image.

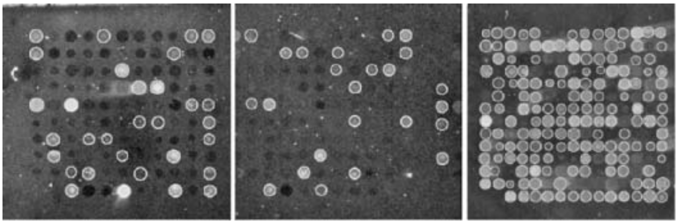


**Figure 28.4** cDNA Microarray spot segmentation results.

threshold. Once the subregion is thresholded, a best-fit circle is computed for the final spot segmentation. Figure 28.4 presents some spot segmentation examples for blocks of different spot densities and qualities from different microarray images.

## 28.4 DATA PROCESSING

Once the spots in a microarray image are extracted, the intensity value of each spot can be obtained and the log ratio (*i.e.*, $M = \log_2 R/G$), which indicates the differential expression of the two DNA samples, can be computed. However, before the expression data can be subjected to further analysis, some preprocessing of the raw intensity data is needed. The data processing steps usually involve (i) background correction, (ii) data normalization, and (iii) data filtering.

### 28.4.1 Background Correction

Background correction aims to remove from the spot's measured intensity a contribution not caused by the specific hybridization of the target to the probe. This contribution could result from nonspecific hybridization and stray fluorescence on the slide. Different approaches, ranging from simple subtraction of local background intensity to sophisticated statistical correction have been proposed [15].

### 28.4.2 Normalization

The purpose of normalization is to adjust for any bias that results from variation in the microarray process rather than from biological differences between the RNA samples. Position variation on a slide may develop because of differences between the print tips on the arrayer, variation over the course of the print run, or nonuniformity in the hybridization. Another common variation is the red-green bias resulting from the differences between the labeling efficiencies of the two dyes, the fluorescent properties of the two dyes, and the scanner settings. It is necessary to normalize the spot intensities before any subsequent analysis is carried out.

The most widely used within-slide normalization method assumes that the red-green bias is constant on the log-scale across the slide. The log ratios are corrected by subtracting a constant $c$ to get the normalized values $M_{\text{norm}} = \log_2(R/G) - c$. The constant $c$ usually is estimated from the mean or the median log ratios value over a subset of the genes assumed to be expressed not differentially [4]. However, the imbalance in the red and green intensities is usually not constant across the spots within and between the slides and can vary according to overall spot intensity, location on the slide, slide origin, and possibly other variables. More sophisticated normalization methods such as loess normalization are available to account for these dependencies [29, 33]. Additionally, housekeeping genes can be used as control spots for normalization.

### 28.4.3 Data Filtering

Not all data extracted from a microarray experiment are useful. Some expression ratios might be unreliable because of the poor quality of the spots and usually are discarded prior to subsequent data analysis. Finally, the expression ratios usually are log transformed (base 2 log) prior to further analysis such as clustering. The log transform renders the $k$-fold change in ratio value symmetrical around the nondifferentially expressed ratio of 1.

## 28.5 MISSING VALUE IMPUTATION

Gene expression microarray experiments usually suffered from the missing values problem. It is not uncommon to find microarray data with up to 90% of genes affected by missing values [22]. Missing values occur for various reasons, including hybridization failures, artifacts on the microarray, insufficient resolution, and image noise and corruption [32]. Missing values have an important implication for subsequent data analysis. For example, the inability of many cluster algorithms to process the missing values means that profiles containing missing values often are discarded. However, instead of ignoring gene expression profiles containing missing values, such missing values often can be estimated from the data.

Reliable estimation of missing values is important. If an erroneous missing value imputation is performed, then gene expressions containing a high number of missing values can be assigned to the wrong cluster in subsequent cluster analysis. The most common method to deal with missing values is simply replacing them with zeros or with the average of the expression profile. Such simple estimation techniques, however, made very crude use of the available knowledge within the data. Current research demonstrated that missing values estimation can be improved significantly by exploiting the correlation between data. Many advanced missing value imputation algorithms have been proposed recently, such as the $K$-nearest neighbors method (KNNIMPUTE), the singular value decomposition method (SVDIMPUTE) [31], least-square imputation (LSIMPUTE) [2], Bayesian principle component analysis (BPCA) [21], local least-square imputation (LLSIMPUTE) [14], and Gaussian mixture imputation (GMCIMPUTE) [22]. These algorithms can perform well depending on the characteristics of the data. For example, KNNIMPUTE performs better on nontime series data or noisy time series data, whereas SVDIMPUTE works well on time series data with a low noise level and with a strong global correlation

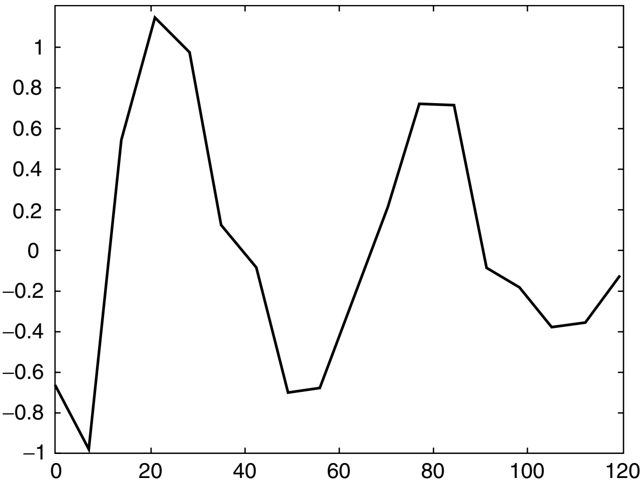


**Figure 28.5** The gene expression profile of Smc3 in Spellman *et al.*'s experiment. The synchronization loss is significant.

structure. LLSIMPUTE has the best performance when a strong local correlation exists in the data. The BPCA method is suitable when a global structure is dominant in the data. LSIMPUTE tries to exploit adaptively both the global and the local structure in the data. However, all these algorithms do not consider biological constraints related to the microarray experiments.

For microarray data, the phenomenon of synchronization loss in the gene expression experiments for time series data has been known for some time [1]. Cyclic systems, such as the cell cycle [30] and circadian clock [23] play a key role in many biological processes. Microarray experiments that study these systems usually are carried out by synchronizing a population of cells. Synchronization is achieved by first arresting cells at a specific biological life point and then by releasing cells from the arrest so that all cells are at the same point when the experiment begins [12, 30]. However, even if cells are synchronized perfectly at the beginning of the experiment, they do not remain synchronized forever [1]. For example, yeast cells seem to remain relatively synchronized for two cycles [30], whereas wild type human cells lose their synchronization very early [28] or halfway through the first cycle depending on the arresting method. This causes the peak expression value to be lower and the lowest expression value to be higher in subsequent cycles for most cyclic genes. A typical gene expression profile with synchronization loss is given in Figure 28.5. We see that because of the loss of synchronization, the peaks and troughs decrease in magnitude with time. As a consequence, the average signal power in the successive cycle decreases significantly. Table 28.1 quantitatively shows the decrease of the average signal energy of four datasets in Spellman *et al.*'s experiment.

In [11], we propose a set theoretic framework based on the projection onto convex sets (POCS) for missing data imputation called POCSIMPUTE, which takes into

**Table 28.1 The decrease of average signal energy resulting from synchronization loss for four datasets in Spellman *et al.*'s experiment. Note that because the signal of Elutriation is available only for one cycle, we compare the average signal energy for the first half cycle and the second half cycle**

| Dataset | No. of Sampling Points | No. of Complete Genes | Average Signal Energy for First Cycle | Average Signal Energy for Second Cycle |
|---|---|---|---|---|
| CDC28 | 17 | 1383 | 352.78 | 288.73 |
| CDC15 | 24 | 4381 | 846.28 | 834.57 |
| Alpha factor | 18 | 4489 | 474.58 | 306.24 |
| Elutriation | 14 | 5766 | 898.48 | 435.32 |

consideration this biological phenomenon of synchronization loss. POCS allows us to incorporate different types of *a priori* knowledge about missing values into the estimation process. The main idea of POCS is to formulate every piece of prior knowledge into a corresponding convex set and then use a convergence-guaranteed iterative procedure to obtain a solution in the intersection of all these sets. Our imputation method captures localized gene-wise correlation in the gene expression data by constructing a convex set based on local least-square regression using the $K$-most correlated genes. We capture the array-wise variation by using the principal component analysis (PCA) method. In the PCA method, the dominant array-wise variation of the entire dataset is summarized by a few principle components, which can be viewed as representing independent cellular states across all genes. Finally, to take advantage of the phenomenon of synchronization loss in microarray experiments, we constrain the vector length of the missing values to be bound by the vector length of the observed values within the same cycle.

The POCS method provides a very flexible framework to incorporate all *a priori* information to get an optimal solution. Regardless of whether it is a consistent problem, the convergence of the algorithm is guaranteed. For gene expression data, which is noisy and with imprecise prior information, this tolerance to imprecision is very important. Another useful feature of our POCSimpute algorithm is its adaptivity in finding a good solution. Suppose we have correlation information between genes and between samples, and these two pieces of information are modeled as two convex sets $C_u$ and $C_v$, respectively. In one dataset, the first piece of information may be more reliable than the second. In another dataset, it may be the opposite. This situation is depicted in Figure 28.6. When the information is more reliable, the corresponding convex set will be smaller in range. Because POCS always converges to the intersection, the final solution always will be dominated by the smaller set while still satisfying the constraint imposed by the less reliable set. In this manner, a good solution that makes a wise trade-off between different prior information can be obtained. Experiments have shown that our algorithm can achieve a significant reduction of error compared with some available algorithms on the test data.

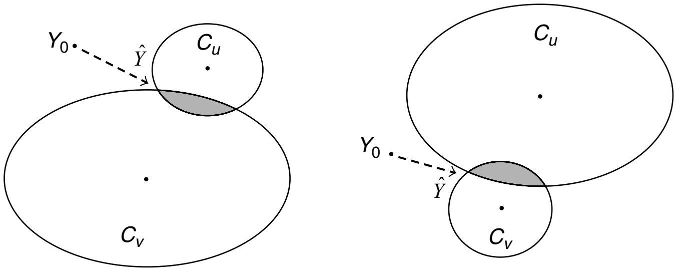


**Figure 28.6** Left: In a data dominated by gene-wise correlation, the final solution is dominated by $C_u$. Right: In a data dominated by array-wise correlation, the final solution is dominated by $C_V$.

## 28.6 TEMPORAL GENE EXPRESSION PROFILE ANALYSIS

Time-series whole-genome expression data are a particularly valuable source of information because they can describe a dynamic biological process such as the cell cycle or metabolic process [7, 30]. They allow the determination of regulatory relationships between the expressions of different genes. Such a relationship could lead to a better understanding of the gene networking process within a cell.

A pairwise comparison of gene expression profiles has been proposed to identify pairs of genes that have direct regulatory relationships. Among the various pairwise comparison approaches, a correlation-based method is perhaps the most popular one. This method determines whether two genes have a regulatory relationship by checking the global similarity between their expression profiles using the Pearson correlation measure [9]. However, it does not take into account the fact that there is often a time delay before the regulator gene product can exert its influence on the target gene. Such a time delay can degrade significantly the performance of the method. The correlation method also strongly favors global similarity over more localized similarities resulting from conditional regulatory relationships.

If the expression of gene A varies periodically at a constant frequency, then we expect the expression of gene B to be varying more or less at the same frequency. This frequency of variation, however, may not be observed easily from the two time series expression profiles as a result of noise and other factors. In addition, if gene B is under the influence of both gene A and gene C (a “two-regulating-one” situation) and if the expression profiles of these influencing genes are varying at different frequencies, then the relationship between gene A and gene B may not be observed easily from their time series profiles. This would cause problem for a correlation-based similarity comparison.

In [34], we propose a spectral component correlation approach for measuring the correlation between time-series expression data and use the results to infer the potential regulatory relationships between genes. Our technique summarizes the essential features of an expression pattern by means of a frequency spectrum estimated by autoregressive modeling. Specifically, the pattern $x[n]$ is decomposed into a set of damped sinusoids of different frequencies,

$$x[n] = \sum\nolimits_{i=1}^{M} x_i[n] = \sum\nolimits_{i=1}^{M} \alpha_i \exp(\sigma_i n) \cos(\omega_i n + \phi_i) \qquad (28.2)$$

so that each sinusoid $x_i[n]$ can be considered separately during the analysis. The parameters $\alpha_i$, $\sigma_i$, $\omega_i$, and $\phi_i$ are the amplitude, damping factor, normalized frequency, and phase angle, respectively, of spectral component $i$. The correlation of $x[n]$ with $y[n]$ then can be reformulated as a sum of the component-wise correlations between each spectral component,

$$x[n] \circ y[n] = \sum_i \sum_j \sqrt{\frac{E_{x_i} E_{y_j}}{E_x E_y}} x_i[n] \circ y_j[n] \qquad (28.3)$$

where $\circ$ denotes the correlation operation and $E$ represents either the total energy of a sequence or the energy of a particular component. Such a component-wise correlation provides more insight into the regulatory relationship. For instance, for the "two-regulating-one" situation, correlation between the expression profiles of gene A and gene B may not be strong enough to suggest their relationship because of the presence of spectral components in gene B induced by gene C. However, the spectral components of gene B resulting from gene A would exhibit strong correlations to gene A's expression profile.

Transcriptional regulation can involve activation or inhibition. In the activation process, the product of gene A affects the transcription process of gene B such that the production rate for gene B increases. On the other hand, the inhibition process involves gene A's product decreasing the production of gene B. In [34], we use the spectral component correlation algorithm to analyze the alpha-synchronized yeast cell-cycle dataset from [30]. We could detect many regulatory pairs that were missed by the traditional correlation method as a result of a weak correlation value. Figure 28.7 shows two known activation pairs. The first one involves genes YLR256W and YPR191W, and the second one involves genes YBR240C and YPL258C. As is shown, the two genes in each regulatory pair do not have similar expression patterns, and their correlation coefficients are –0.1491 and –0.1127 for the first and second pairs, respectively. However, the lowest frequency components in each pair have closely matched spectral characteristics. These lowest frequency components identify the general variations for the profiles, and the time lags between the activators and activatees clearly are revealed.

The spectral component correlation method allows us to identify strongly oscillatory but time-shifted expression pairs by using only the spectral magnitude information and ignoring the phase information. Figure 28.8(a) shows the expression profiles and spectrums for an activation pair involving genes YAL040C and YER111C. These two patterns strongly oscillate at around 0.76 rad/s but still have a relatively low correlation value of –0.3885 because of the time lag between them and because of other unmatched components. The spectral magnitude plot (the top plot in

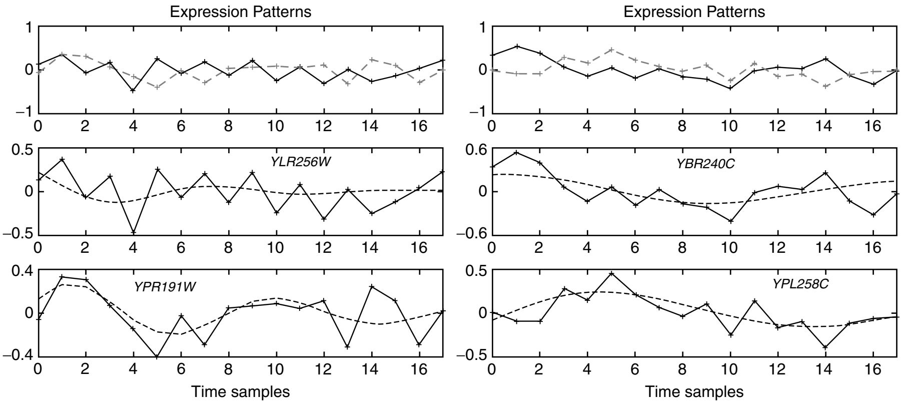


**Figure 28.7** Left: Known activation regulation involving genes YLR256W and YPR191W. Right: Known activation regulation involving genes YBR240C and YPL258C. In both the left and right panels, the top graph shows the gene expression profiles for the regulatory pairs, whereas the middle and bottom graphs show each of the expression profiles with their corresponding lowest frequency components.

Figure 28.8(c)) shows that their dominant components are closely matched with each other. Note that the spectral component for gene YAL040C at frequency of 3.1416 rad/s is not considered dominant because of its large decay factor (see bottom plot in Figure 28.8(c)). Figure 28.8(b) shows an inhibition regulation involving genes YBR049C and YGR254W. A careful examination of the dominant component's phase angle in these gene pairs suggests that the activatee has a phase lag within 0° to 180° relative to the activator's phase angle, whereas the inhibitee has a phase-lead within 0° to 180° relative to the inhibitor's phase angle.

The spectral component correlation method allows us to neglect certain irrelevant components that otherwise may corrupt the correlation between the two patterns. Numerous known regulations with weak correlations are caused by such irrelevant components. For example, the components at 0.7248 rad/s for gene YAL040C and at 0.8066 rad/s for gene YER111C (see Table 28.2) clearly dominate over other components. The component-wise correlation with phase alignment using just this component is 0.7665. Compared with the original correlation value of –0.3885, the component-wise correlation strongly suggests the similarity between the two patterns.

When the component-wise correlation analysis is applied to all 439 known regulations in the alpha-synchronized yeast cell-cycle dataset, the results indicated that 223 out of 343 activation pairs and 55 out of 96 inhibition pairs have a component-wise correlations score greater than 0.5 (see Table 28.3). In contrast, the traditional correlation method only can detect 36 activation pairs and 5 inhibition pairs. Several of visually dissimilar expression pairs are observed to have very similar dominant frequency components. For example, among those 307 pairs with traditional correlation coefficients of less than 0.5, 196 of them have component-wise correlation coefficients greater than 0.5. Furthermore, 60 out of these 196 pairs have component-wise

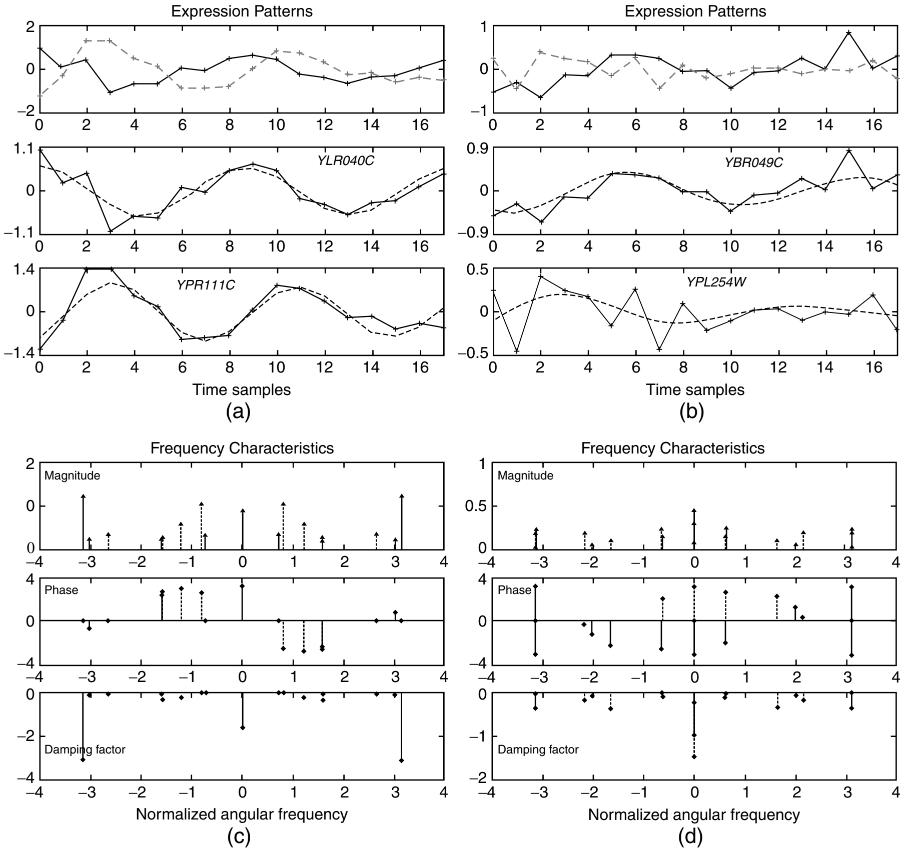


**Figure 28.8** (a) Known activation regulation involving genes YAL040C and YER111C. The dominant frequency components for these two genes are plotted in the middle and the bottom graphs. Although they both strongly oscillate at a frequency around 0.76 rad/s, the time lag together with other unmatched components makes them have a low correlation value of –0.3885. (b) Known inhibition regulation involving genes YBR049C and YGR254W with a correlation coefficient of –0.3226. The dominant frequency components for this gene pair are plotted in the middle and bottom graphs. (c) Frequency characteristics for the regulatory gene pair YAL040C and YER111C. The dominant frequencies for these two profiles are 0.7248 rad/s and 0.8066 rad/s, respectively. (d) Frequency characteristics for the regulatory gene pair YBR049C and YGR254W. The dominant frequencies are 0.6395 rad/s and 0.6271 rad/s, respectively.

correlation coefficients greater than 0.9, and the expression patterns in each of these pairs strongly oscillate at almost identical frequencies. The spectral component correlation method allows the hidden component-wise relationships between two expression profiles to be revealed, which otherwise are hidden in the traditional correlation analysis.

A strength of the spectral component correlation method is its ability to detect regulatory relationships involving multiple genes. For regulations involving a single gene being regulated simultaneously by two or more genes with different

**Table 28.2 Estimated frequency components for expression profiles of genes YAL040C and YER111C in which components with strong correlation are highlighted**

| | YAL040C | | | | YER111C | | | |
|---|---|---|---|---|---|---|---|---|
| $i$ | $\sigma_i$ | $\omega_i$ | $\alpha_i$ | $\varphi_i$ | $\sigma_i$ | $\omega_i$ | $\alpha_i$ | $\varphi_i$ |
| 1 | –1.5993 | 0.0000 | 0.8670 | 3.1416 | –0.0203 | 0.8066 | 0.5167 | –2.5294 |
| 2 | –0.0047 | 0.7248 | 0.3177 | –0.0226 | –0.2575 | 1.2087 | 0.2854 | –2.8483 |
| 3 | –0.0729 | 1.5913 | 0.1697 | –2.3890 | –0.3592 | 1.5766 | 0.1318 | –2.6949 |
| 4 | –0.1313 | 3.0256 | 0.2068 | 0.6806 | –0.0869 | 2.6468 | 0.1626 | –0.0445 |
| 5 | –3.1281 | 3.1416 | 1.1970 | 0.0000 | — | — | — | — |

expression frequencies, we can identify them by checking for regulators' frequencies from the expression profile of the gene being regulated. Figure 28.9 shows two known activation regulations with the common gene YPR120C as an activatee. It reveals that the first regulation has its expression profiles correlated at a frequency of around 1.48 rad/s, whereas the second regulation has its profiles correlated at around 0.76 rad/s. Table 28.4 lists eight "$n$-regulating-one" activation sets identified using the spectral component correlation method.

To see how the spectral component correlation method can be used to infer causal relationship, the genes YBR240C and YAL040C are used as references to find all

**Table 28.3 Results for the two correlation methods applied to all 439 known regulatory pairs in the alpha-synchronized yeast cell-cycle dataset. (a) Statistics for the 343 activation pairs. (b) Statistics for the 96 inhibition pairs**

| a | Traditional Correlation < 0.5 | Traditional Correlation > 0.5 | Total |
|---|---|---|---|
| Component-wise Correlation < 0.5 | 111 | 9 | 120 |
| Component-wise Correlation > 0.5 | 196 | 27 | 223 |
| Total | 307 | 36 | 343 |

| b | Traditional Correlation <–0.5 | Traditional Correlation >–0.5 | Total |
|---|---|---|---|
| Component-wise Correlation >–0.5 | 1 | 40 | 41 |
| Component-wise Correlation <–0.5 | 4 | 51 | 55 |
| Total | 5 | 91 | 96 |

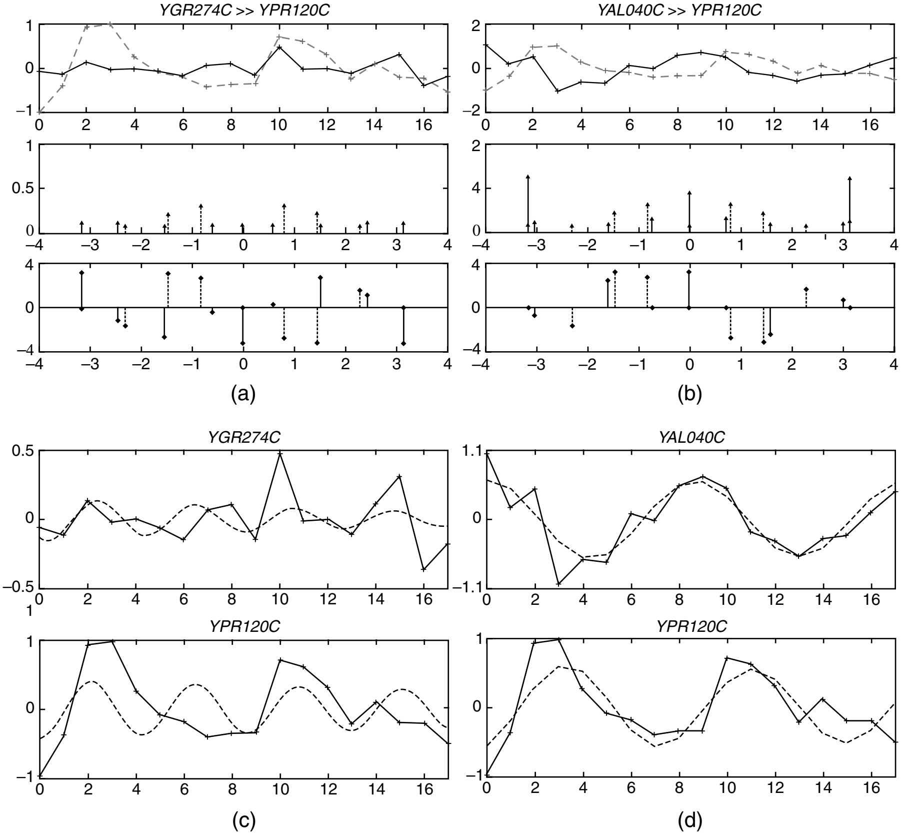


**Figure 28.9** Two activation regulations with gene YPR120C as an activatee. (a) Activation regulation with gene YGR274C as an activator. (b) Activation regulation with gene YAL040C as an activator. (c) Correlated frequency components for the first pair. (d) Correlated frequency components for the second pair.

other genes in the Filkov's dataset [10], which has a component-wise correlation coefficient of greater than 0.7. There are 55 out of 288 genes for YBR240C and 59 out of 288 genes for YAL040C that satisfy this threshold. These two sets of genes with their scores are shown in Figure 28.10, and their oscillatory properties clearly are revealed when they are arranged such that their phase is in descending order. Within these genes, one known activation regulation of gene YBR240C is contained in the first set and three for gene YAL040C are contained in the second set. Note that genes below the reference gene have their phases lag by 0° to 180° relative to the reference gene's phase, and they can be considered as potential activated candidates. On the other hand, genes above the reference gene have their phases lead by 0° to 180°, and they can be considered potential inhibited candidates. If we look at the known activatees for the two examples shown in Figure 28.10, then we see that they all are located below their corresponding activators.

**Table 28.4 Various activation regulation sets. Each set contains a common activatee, which has two different correlated frequency components**

| Activator | Activatee | Traditional Correlation | Component-Wise Correlation | Activator Frequency | Activatee Frequency |
|---|---|---|---|---|---|
| YKL109W | YGL167C | 0.2877 | 0.9237 | 0.6505 | 0.6842 |
| YLR433C | YGL167C | 0.1980 | 0.5287 | 1.3502 | 1.6359 |
| YHR079C | YJL034W | –0.6594 | 0.9894 | 0.7063 | 0.7205 |
| YPL085W | YJL034W | 0.2717 | 0.6120 | 1.7581 | 1.9513 |
| YKL109W | YLL041C | 0.3792 | 0.9917 | 0.5339 | 0.5230 |
| YBL021C | YLL041C | 0.2586 | 0.9564 | 1.3748 | 1.3725 |
| YGL237C | YLL041C | –0.4687 | 0.8484 | 0.6456 | 0.5230 |
| YOR358W | YLL041C | 0.3800 | 0.8008 | 1.2639 | 1.3725 |
| YLR182W | YLR286C | –0.1208 | 0.8984 | 1.1082 | 1.0378 |
| YLR071C | YLR286C | 0.0349 | 0.6662 | 0.3324 | 0.4353 |
| YLR131C | YLR286C | –0.2762 | 0.6535 | 0.5338 | 0.4353 |
| YEL009C | YOR202W | 0.0554 | 0.9276 | 1.2653 | 1.1670 |
| YRL082C | YOR202W | 0.6075 | 0.8912 | 0.3199 | 0.3517 |
| YEL009C | YPR035W | –0.3737 | 0.9541 | 1.2653 | 1.2241 |
| YFL021W | YPR035W | –0.2153 | 0.9002 | 0.4095 | 0.3662 |
| YGR274C | YPR120C | 0.4075 | 0.8541 | 1.5266 | 1.4566 |
| YAL040C | YPR120C | –0.4331 | 0.7288 | 0.7248 | 0.8120 |
| YLR256W | YPR191W | –0.1491 | 0.9173 | 0.7762 | 0.7295 |
| YGL237C | YPR191W | –0.7333 | 0.8821 | 0.6456 | 0.7295 |
| YBL021C | YPR191W | –0.2231 | 0.7569 | 1.3748 | 1.4294 |
| YOR358W | YPR191W | 0.0937 | 0.7209 | 0.6227 | 0.7295 |
| YKL109W | YPR191W | 0.2663 | 0.6254 | 0.5339 | 0.7295 |

## 28.7 CYCLIC GENE EXPRESSION PROFILES DETECTION

Oscillation results in genetic and metabolic networks as a result of various modes of cellular regulation. These rhythmic processes occur at all levels of biological organization with widely varying periods. Well-known examples of biological rhythms include cell division [20, 25, 30] and circadian rhythms [6, 27]. Rhythmic cellular processes are regulated by different gene products and can be measured through a series of DNA microarray experiments. If the expression patterns of a group of genes are measured over several time points, then we obtain a time series gene expression profile describing the rhythmic behaviors of the genes under study.

A well-known set of gene expression time series datasets is that of the yeast (*Saccharomyces cerevisiae*) from Spellman *et al.* [30]. In this set of data, the genome-wide mRNA levels for 6178 yeast open reading frames are monitored simultaneously using several different methods of synchronization including an alpha-factor-mediated G1 arrest, which covers approximately two cell-cycle periods with measurements at 7 minute intervals for 119 minutes with a total of 18 time points, a temperature-sensitive cdc15 mutation to induce a reversible M-phase arrest (24 time points taken every 10 minutes covering approximately 3.5 cell-cycle

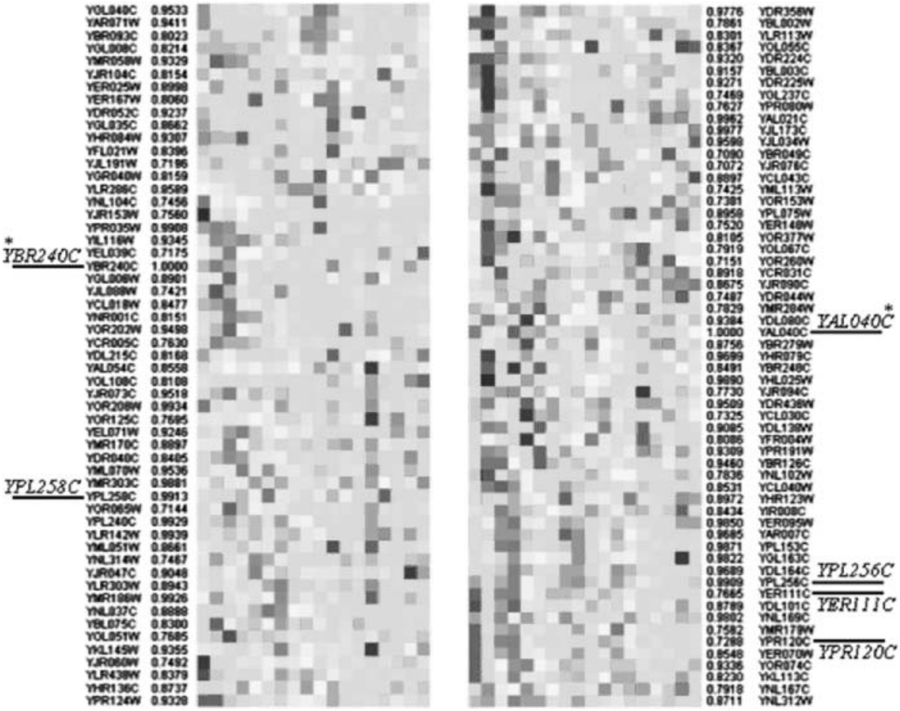


**Figure 28.10** Genes in the Filkov's dataset have component-wise correlation coefficients, relative to gene YBR240C (left) and gene YAL040C (right), greater than 0.7. The known activation regulations with genes YBR240C and YAL040C as activators are highlighted.

periods), a temperature-sensitive cdc28 mutation to arrest cells in G1 phase reversibly (17 time points taken every 10 minutes covering approximately 2 cell-cycle periods), and finally, an elutriation synchronization to produce the elutriation dataset of 14 time points taken every 30 minutes covering approximately 1 cell-cycle period. Figure 28.11 shows some periodic (top panel) and random (bottom panel) profiles from this dataset.

The detection of gene expression profiles that exhibit cyclic behavior is a difficult problem for several reasons. First, a gene expression time series profile usually contains few time points. It is not uncommon to see expression profiles that are less than 10 time points long. Second, the number of cycles within a profile is usually low. For example, the 14-time-point elutriation dataset of Spellman *et al.* [30] contains only one cell-cycle. Third, gene expression data usually contains a lot of missing values, and these missing values usually need to be estimated from the dataset in advance. Fourth, the time points do not need to be spaced at regular intervals, resulting in the problem of detecting periodicity in unevenly sampled time series data. Finally, gene expression data is notoriously noisy. Recently, we have proposed several effective computational techniques to detect periodic expression profiles based on: (i) singular spectrum analysis (SSA) and autoregressive (AR)-based spectral

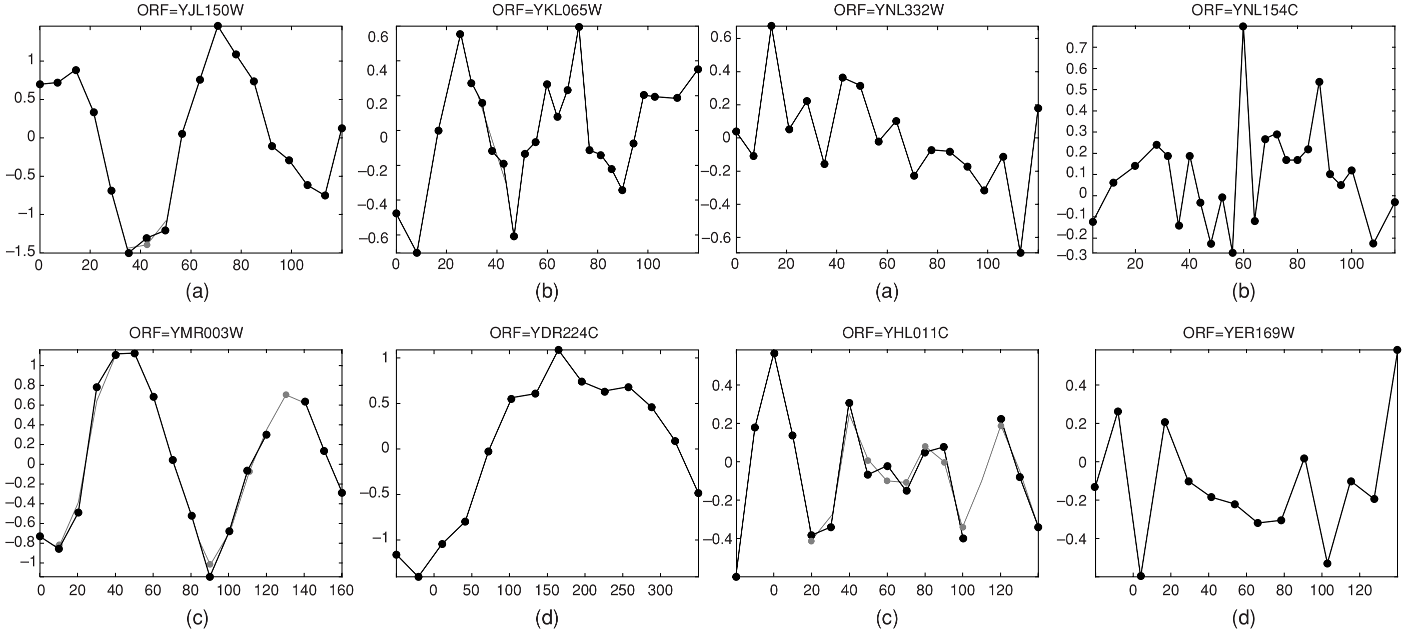


**Figure 28.11** Left panel: highly periodic expression profiles. Right panel: random profiles from yeast datasets of Spellman *et al*. [30]. Profiles (a) through (d) correspond to the alpha, cdc15, cdc28, and elutriation datasets, respectively. The thin curves in the figure are the interpolated profiles with the missing values filled in. The x-axis shows the time points, and the y-axis shows the measured expression values.

estimation (SSA-AR), (ii) spectral estimation of short, unevenly sampled profiles by signal reconstruction, and (iii) statistical hypothesis testing for periodic signal detection. These techniques have enabled us to identify reliably short, periodically expressed gene profiles that are involved in the cell-cycle process.

### 28.7.1 SSA-AR Spectral Estimation

In [8], we proposed a parametric spectral estimation technique for short time series profiles based on SSA and AR modeling. The AR model for a time series $s(n)$ is given by

$$s(n) = -\sum_{p=1}^{P} a_p s(n-p) + u(n)$$

where $a_p$ is the AR coefficients, $P$ is the order of the AR model, and $u(n)$ is a white-noise sequence. The AR model-based power spectrum estimation allows better frequency resolution to be obtained by fitting a relatively high order AR model to the data sequence. However, the AR spectrum is sensitive to noise. When the signal-to-noise ratio is low, the accuracy of the parameter estimation would be reduced substantially. Using a higher order AR model to improve the frequency resolution also would induce the appearance of spurious peaks. To remedy this problem, we proposed preprocessing the profiles using SSA to extract the underlying trend from the short and noisy time series.

SSA performs a singular value decomposition (SVD) on the trajectory matrix obtained from the original time series. Let each expression profile be a time series $\{s_1, s_2, \ldots, s_n, \ldots, s_N\}$, and the trajectory matrix $X_{M,K}$ can be obtained from the original series by sliding a window of length $M(M \leq N/2)$, $K = N - M + 1$,

$$X_{M,K} = (x_{ij} = s_{i+j-1}) = \begin{bmatrix} s_1 & s_2 & \cdots & s_K \\ s_2 & s_3 & \cdots & s_{K+1} \\ \vdots & \vdots & \cdots & \vdots \\ s_M & s_{M+1} & \cdots & s_N \end{bmatrix} \tag{28.4}$$

The singular values of the SVD of the matrix $R = XX^T$ then can be grouped into two separate components: a trend component and a noise component. With the proper selection of singular values, the trend curve that represents the dominant spectral component can be reconstructed from the original expression profile. Using SSA and AR, periodic profiles can be detected as the reconstructed profiles that exhibit strong dominant frequency in the AR spectrum (*i.e.*, those that satisfy a power ratio threshold).

The SSA-AR spectral estimation-based technique is applied to the expression data of the IDC of *Plasmodium falciparum*. The data contains the expression profiles of 5080 oligonucleotides measured at 46 time points spanning 48 hours during the IDC with one-hour time resolution for the HB3 strain [3]. Figure 28.12 shows the

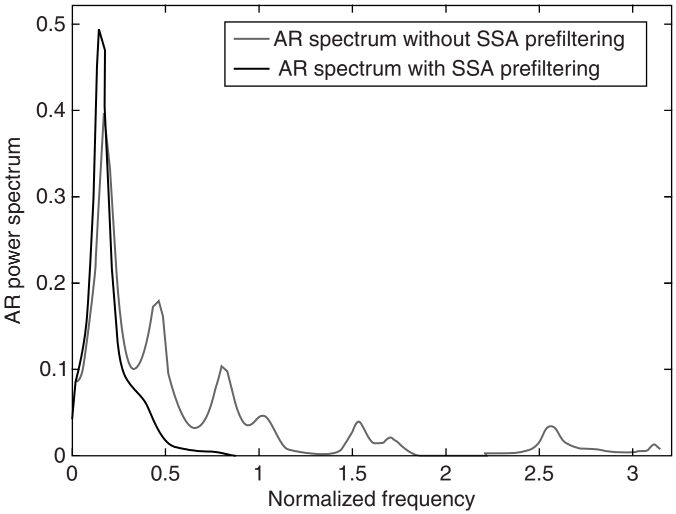


**Figure 28.12** The AR spectra of the expression profile of DHFR-TS with and without SSA filtering.

AR spectrum of one oligonucleotide dihydrofolate reductase-thymidylate synthase (DHFR-TS) in this dataset before and after SSA preprocessing. One can see that spurious spectra are suppressed after SSA filtering. By using this method, we could detect 4496 periodic profiles. Compared with Bozdech *et al.* [3], an additional 777 periodic oligonucleotides are detected using our algorithm.

Because the function of a gene is related to the initial phase of its expression profile, we have ordered the expression profiles of the 4496 periodic oligonucleotides according to their peak time points of expression profiles as in Figure 28.13. The phaseogram shows a continuous cascade of gene expressions, which correspond to the developmental stages throughout the IDC (*i.e.*, ring, trophozoite, and schizont stages). According to the sharp transitions of ring-to-trophozoite (at the 17-h time point), trophozoite-to-schizont (at the 29-h time point), and schizont-to-rings stages (at the 45-h time point), the 4496 periodic genes could be categorized into four stages based on the peak time points of their expression profiles. Table 28.5 compares the classification results of the oligonucleotides assigned to these stages by our method with those in Bozdech *et al.* [3]. As is shown, more genes can be identified using our method.

### 28.7.2 Spectral Estimation by Signal Reconstruction

In many microarray time series data, the microarray experiments are not carried out at regular sampling intervals [5, 30]. Moreover, missing values are a common occurrence in microarray data. Time series profiles with missing values can be viewed as unevenly sampled. The unevenly sampled profiles make spectral analysis a challenging task. In [17], we proposed a new spectral estimation algorithm for unevenly

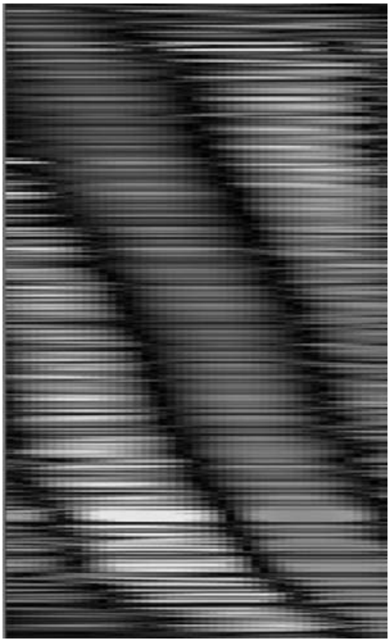


**Figure 28.13** The phaseogram of the transcriptome of the IDC of *P. falciparum*. 4496 genes are ordered along the y-axis in the order of the time of their peak expression.

sampled gene expression data. The method is based on signal reconstruction in a shift-invariant signal space in which a direct spectral estimation procedure is developed using the B-spline basis.

Let $V(\phi)$ be the shift-invariant signal space

$$V(\phi) = \{f : f(x) = \sum\nolimits_{k \in Z} c_k \phi(x - k)\} \tag{28.5}$$

where the coefficients $\{c_i\}$ are related to the choice of basis function $\phi$. If $\phi$ is chosen to be the B-spline interpolating functions, then we obtain an explicit formulation of

**Table 28.5 Classification Results of Oligonucleotides in Three Different Stages**

| Stages | Our Method | Bozdech *et al.* [3] |
|---|---|---|
| Ring/early trophozoite | 1970 | 1563 |
| Trophozoite/early schizont | 1524 | 1296 |
| Schizont | 709 | 625 |
| Early ring | 293 | 235 |

the power spectrum density (PSD) as

$$P_{xx}(\omega) = \frac{1}{A_2 - A_1}\left|\sum_{k=A_1-\Omega+1}^{A_2+\Omega-1} c_k e^{-i2\pi\omega k}\widehat{\phi}(\omega)\right|^2 \tag{28.6}$$

where $\hat{\phi}(\omega) = \left[\sin(\pi\omega)\big/\pi\omega\right]^{N+1}$. Our method allows the PSD of an unevenly sampled signal to be computed directly. Because the periodogram of a microarray time series profile from a periodically expressed gene must contain a peak corresponding to its dominant frequency, we can perform a statistical test on the PSD to determine whether a time series profile is periodic or random. We applied the method on the gene expression dataset of *P. falciparum* and showed that it gives a good estimate of the number of periodic genes.

### 28.7.3 Statistical Hypothesis Testing for Periodic Profile Detection

The problem of deciding whether a time series is random or periodic can be cast as a statistical decision problem using hypothesis testing. Given a time series $y$ of length $N$, the periodogram $I(\omega)$ is first computed as follows:

$$I(\omega) = \frac{1}{N}\left|\sum_{n=1}^{N} y_n e^{-j\omega n}\right|^2 \qquad \omega \in [0, \pi] \tag{28.7}$$

and $I(\omega)$ is evaluated at the discrete normalized frequencies $\omega_l = 2\pi l/N, \quad l = 0, 1, \ldots, a$, where $a = [(N-1)/2]$ and $[x]$ denotes the integer part of $x$. If a time series has a significant sinusoidal component with frequency $\omega_k$, then the periodogram will exhibit a peak at that frequency $\omega_k$. An exact test of the significance of the spectral peak can be done using the Fisher $g$-statistic $g = \max_l I(\omega_l)\big/\sum_{l=1}^{a} I(\omega_l)$.

Under the Gaussian noise assumption, the exact distribution of the $g$-statistic under the null hypothesis (that the spectral peak is insignificant) is given by

$$P(g > x) = \sum_{k=1}^{b} (-1)^{k-1}\frac{a!}{k!(a-k)}(1 - kx)^{a-1} \tag{28.8}$$

where $b$ is the largest integer less than $1/x$ and $x$ is the observed value of the g-statistic. Equation (28.28.8) yields a $p$-value that allows us to test whether a given time series behaves like a random sequence. A large value of $g$ indicates a strong periodic component and leads us to reject the null hypothesis.

Although the exact distribution of the Fisher $g$-statistic is available analytically, we found that care must be taken when applying it in practice. Figure 28.14 shows the exact distribution and the empirical distribution for signal length $N = 10$. The deviation from the exact distribution can be seen clearly. Nevertheless, for the larger value of $N$ (*i.e.*, $N > 40$), no significant deviation can be observed.

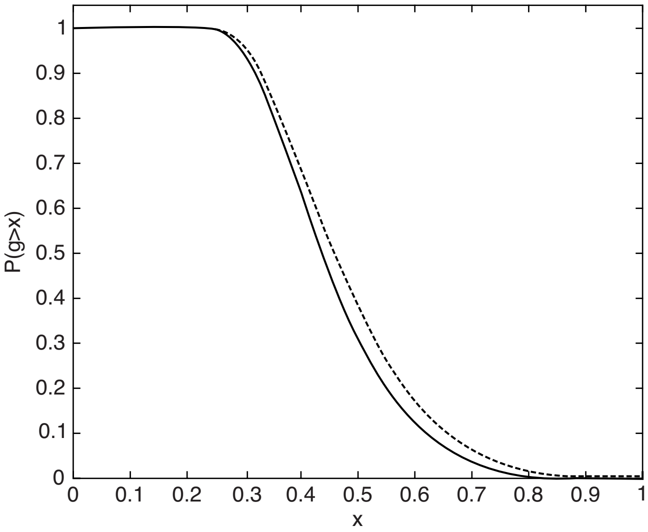


**Figure 28.14** Empirically computed distribution (dashed curve) versus theoretical distribution (solid curve) for a short length signal ($N = 10$).

In [18], we performed a series of experiments with simulated signals to investigate the statistical power of the Fisher test as a function of noise distribution, signal length, signal-to-noise ratio, and the false discovery rate. We found that the deviation from the theoretical null distribution can be significant when the signal length is shorter than 40 time points. Moreover, when the signal does not cover an integer number of periods, a significant drop in the statistical power of the test was observed. In this case, a much longer signal is needed for the test to return a reliable result. These findings indicate that in high likelihood, the number of periodic gene expression profiles can be underestimated severely for a short length signal ($\ll$40 time points) as is the case for many publicly available gene expression datasets. Although our study shows that the Fisher test may be unreliable for a short signal, the Fisher $g$-statistic, on the other hand, has been observed to provide a useful ranking of periodic signals. Strongly periodic signals are found to rank highly, whereas random sequences have a low ranking. In [17], we use this ranking to discover the periodic gene expression profiles in the *P. falciparum* dataset by analyzing the trend of the sorted $g$-statistic (see Figure 28.15) and showed that the number of periodic profiles in the complete dataset should be around 3700 to 4000, a result in agreement with several published results [3, 13].

## 28.8 SUMMARY

Microarray technology has made possible the profiling of gene expressions of the entire genome in a single hybridization experiment. By studying the expression

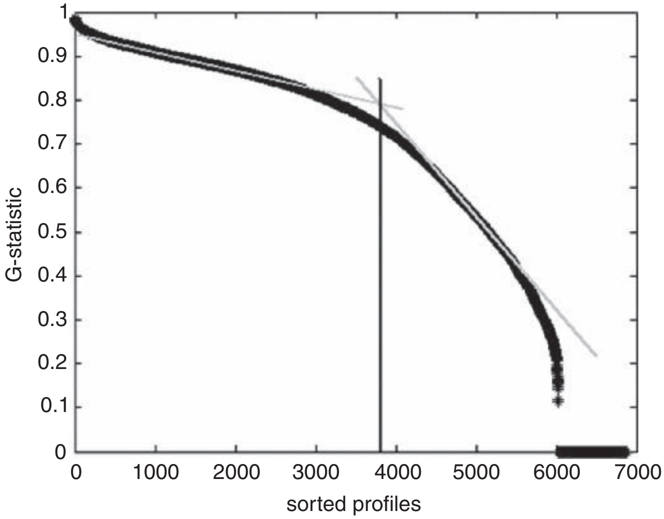


**Figure 28.15** Sorted *g*-statistic values of *P. falciparum*. There is a change in the trend of the ranked *g*-statistic values at around the 4000 sorted profiles, indicating that two classes of profiles (*i.e.*, periodic/aperiodic) are present in the dataset.

profiles of many genes, scientists can gain important insights into gene functions and their interaction in many cellular processes. In this chapter, we present an overview of microarray technology, microarray image analysis and data extraction, and time series gene expression profile analysis. First, we give a brief description of microarray technology and describe how the expression ratio data can be extracted from microarray images. We then discuss how the ratio data are processed prior to subsequent analysis. Because gene expression data usually suffer from a missing value problem, missing value imputation is an important preprocessing step in gene expression data analysis. We describe an imputation technique called POCSIMPUTE that can use gene-wise and array-wise correlation as well as the biological phenomenon of synchronization loss in microarray experiments. Temporal gene expression profiles obtained from performing a series of microarray experiments in discrete time points are valuable in the study of dynamic biological process such as the cell cycle or gene regulation. We describe our work in detecting gene regulatory relationships based on pair-wise spectral component correlation analysis of time series gene expression profiles. Finally, we present several approaches that we have developed in detecting cyclic gene expression profiles.

## ACKNOWLEDGMENTS

We are grateful to Professor Richard Mott for his valuable comments on the manuscript, and to Professor Mourad Elloumi for some interesting discussions.

## REFERENCES

1. Z. Bar-Joseph, S. Farkash, D.K. Gifford, I. Simon, and R. Rosenfeld. Deconvolving cell cycle expression data with complementary information. *Bioinformatics*, 20:i23–i30, 2004.
2. T.H. Bø, B. Dysvik, and I. Jonassen. LSimpute: Accurate estimation of missing values in microarray data with least squares method. *Nucleic Acids Res*, 32:e34, 2004.
3. Z. Bozdech, M. Llinas, B.L. Pulliam, E.D. Wong, J.C. Zhu, and J.L. DeRisi. The transcriptome of the intraerythrocytic developmental cycle of Plasmodium falciparum. *PLoS Biology*, 1:1–16, 2003.
4. Y. Chen, E.R. Dougherty, and M.L. Bittner. Ratio-based decisions and the quantitative analysis of cDNA microarray images. *J. Biomed Optic*, 2:364–374, 1997.
5. S. Chu, J.L. DeRisi, M.B. Eisen, J. Mulholland, D. Botstein, P.O. Brown, I. Herskowitz. The transcriptional program of sporulation in budding yeast. *Science*, 282:699–705, 1998.
6. S.K. Crosthwaite. Circadian clocks and natural antisense RNA. *FEBS Lett*, 567:49–54, 2004.
7. J.L. DeRisi, V.R. Lyer, and P.O. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278:680–686, 1997.
8. L. Du, S. Wu, A.W.C. Liew, D.K. Smith, and H. Yan. Spectral analysis of microarray gene expression time series data of Plasmodium Falciparum. *Int J Bioinform Res Appl*, 4(3):337–349, 2008.
9. M.B. Eisen, P.T. Spellman, P.O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc Natl Acad Sci U S A*, 95:14863–14868, 1998.
10. V. Filkov, S. Skiena, and J. Zhi. Analysis techniques for microarray time series data. *J Comp Biol*, 9(2):317–330, 2002.
11. X. Gan, A.W.C. Liew, and H. Yan. Microarray missing data imputation based on a set theoretic framework and biological consideration. *Nucleic Acids Res*, 34(28.5):1608–1619, 2006.
12. A.P. Gasch, P.T. Spellman, C.M. Kao, O. Carmel-Harel, M.B. Eisen, G. Storz, D. Botstein, and P.O. Brown. Genomic expression programs in the response of yeast cells to environmental changes. *Mol Biol Cell*, 11:4241–4257, 2000.
13. E.F. Glynn, J. Chen, A.R. Mushegian. Detecting periodic patterns in unevenly spaced gene expression time series using Lomb–Scargle periodograms. *Bioinformatics*, 22:310–316, 2006.
14. H. Kim, G.H. Golub, and H. Park. Missing value estimation for DNA microarray gene expression data:local least squares imputation. *Bioinformatics*, 21:187–198, 2005.
15. C. Kooperberg, T.G. Fazzio, J.J. Delrow, and T. Tsukiyama. Improved background correction for spotted DNA microarrays. *J Comp Biol*, 9(28.1):55–66, 2002.
16. A.W.C. Liew, H. Yan, and M. Yang. Robust adaptive spot segmentation of DNA microarray images. *Pattern Recogn*, 36(28.5):1251–1254, 2003.
17. A.W.C. Liew, J. Xian, S. Wu, D. Smith, and H. Yan. Spectral estimation in unevenly sampled space of periodically expressed microarray time series data. *BMC Bioinformatics*, 8:137, 2007.
18. A.W.C. Liew, N.F. Law, X.Q. Cao, and H. Yan. Statistical power of Fisher test for the detection of short periodic gene expression profiles. *Pattern Recogn*, 42(28.4):549–556, 2009.

19. D.J. Lockhart and E.A. Winzeler. Genomics, gene expression and DNA arrays. *Nature*, 405:827–846, 2000.

20. J.M. Mitchison. Growth during the cell cycle. *Int Rev Cytol*, 226:165–258, 2003.

21. S. Oba, M. Sato, I. Takemasa, M. Monden, K. Matsubara, and S. Ishii. A Bayesian missing values estimation method for gene expression profile data. *Bioinformatics*, 19:2088–2096, 2003.

22. M. Ouyang, W.J. Welsh, and P. Georgopoulos. Gaussian mixture clustering and imputation of microarray data. *Bioinformatics*, 20:917–923, 2004.

23. S. Panda, M.P. Antoch, B.H. Miller, A.I. Su, A.B. Schook, M. Straume, P.G. Schultz, S.A. Kay, J.S. Takahashi, and J.B. Hogenesch. Coordinated transcription of key pathways in the mouse by the circadian clock. *Cell*, 109:307–320, 2002.

24. A.C. Pease, D. Solas, E.J. Sullivan, M.T. Cronin, C.P. Holmes, and S.P. Fodor. Light-generated oligonucleotide arrays for rapid DNA sequence analysis. *PNAS*, 91:5022–5026, 1994.

25. G. Rustici, J. Mata, K. Kivinen, P. Lió, C.J. Penkett, G. Burns, J. Hayles, A. Brazma, P. Nurse, and J. Bähler. Periodic gene expression program of the fission yeast cell cycle. *Nat Genet*, 36:809–817, 2004.

26. M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270:467–470, 1995.

27. U. Schibler and F. Naef. Cellular oscillators: Rhythmic gene expression and metabolism. *Cur Opin Cell Biol*, 17(28.2):223–229, 2005.

28. K. Shedden and S. Cooper. Analysis of cell-cycle-specific gene expression in human cells as determined by microarrays and double-thymidine block synchronization. *Proc Natl Acad Sci U S A*, 99:4379–4384, 2002.

29. G.K. Smyth and T.P. Speed. Normalization of cDNA microarray data. *Methods*, 31:265–273, 2003.

30. P.T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen, P.O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Mol Biol Cell*, 9:3273–3297, 1998.

31. O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R.B. Altman. Missing values estimation methods for DNA microarrays. *Bioinformatics*, 17:520–525, 2001.

32. Y.H. Yang, M.J. Buckley, S. Dudoit, and T.P. Speed. Comparison of methods for image analysis in cDNA microarray data. Technical Report 584, Department of statistics, UC Berkeley, 2000.

33. Y.H. Yang, S. Dudoit, P. Luu, D.M. Lin, V. Peng, J. Ngai, and T.P. Speed. Normalization for cDNA microarray data: A robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Res*, 30(28.4):e15, 2002.

34. L.K. Yeung, L.K. Szeto, A.W.C. Liew, and H. Yan. Dominant spectral component analysis for transcriptional regulations using microarray time-series data. *Bioinformatics*, 20(28.5):742–749, 2004.

# 29

# BICLUSTERING OF MICROARRAY DATA

Wassim Ayadi and Mourad Elloumi

## 29.1 INTRODUCTION

One of the main challenges in computational molecular biology is the design of efficient algorithms capable of analyzing biological data, like microarray data. Analysis of gene expression data obtained from microarray experiments can be made through *biclustering*. Indeed, gene expression data are usually represented by a data matrix $M$ (see Table 29.1), where the *i*th row represents the *i*th gene, the *j*th column represents the *j*th condition and the cell $m_{ij}$ represents the expression level of the *i*th gene under the *j*th condition.

In general, subsets of genes are coexpressed only under certain conditions but behave almost independently under others. Discovering such coexpressions can be helpful to discover genetic knowledge such as genes annotation or genes interaction. Hence, it is very interesting to make a simultaneous clustering of rows (genes) and of columns (conditions) of the data matrix to identify groups of rows coherent with groups of columns (*i.e.*, to identify clusters of genes that are coexpressed under clusters of conditions, or clusters of conditions that make clusters of genes coexpress). This type of clustering is called *biclustering* [9]. A cluster made thanks to a biclustering is called *bicluster*. Hence, a bicluster of genes (respectively conditions) is defined with respect to only a subset of conditions (respectively genes). Thus, a bicluster is a subset of genes showing similar behavior under a subset of conditions of the original expression data matrix. Let us note that a gene/condition can belong to more than one bicluster or to no bicluster.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

**Table 29.1 Gene expression data matrix**

| | $Condition_1$ | $\cdots$ | $Condition_j$ | $\cdots$ | $Condition_m$ |
|---|---|---|---|---|---|
| $Gene_1$ | $m_{11}$ | $\cdots$ | $m_{1j}$ | $\cdots$ | $m_{1m}$ |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $Gene_i$ | $m_{i1}$ | $\cdots$ | $m_{ij}$ | $\cdots$ | $m_{im}$ |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| $Gene_n$ | $m_{n1}$ | $\cdots$ | $m_{nj}$ | $\cdots$ | $m_{nm}$ |

In other words, a *bicluster* can be defined as follows: Let $I = \{1, 2, \ldots, n\}$ be a set of indices of $n$ genes, $J = \{1, 2, \ldots, m\}$ be a set of indices of $m$ conditions, and $M(I, J)$ be a data matrix associated with $I$ and $J$. A *bicluster* associated with the data matrix $M(I, J)$ is a couple $(I', J')$ such that $I' \subseteq I$ and $J' \subseteq J$.

Actually, biclustering is a special case of clustering. Indeed, in biclustering, genes are clustered according to their expression levels under several conditions, not necessarily all the conditions. Although in clustering, genes are clustered according to their expression levels under all the conditions. Similarly, in biclustering, conditions are clustered according to the expression levels of several genes not necessarily all the genes.

The *biclustering problem* can be formulated as follows: Given a data matrix $M$, construct a bicluster $B_{\text{opt}}$ associated with $M$ such that:

$$f(B_{\text{opt}}) = \max_{B \in BC(M)} f(B) \tag{29.1}$$

where $f$ is an objective function measuring the *quality* (*i.e.*, degree of coherence) of a group of biclusters and $BC(M)$ is the set of all the possible groups of biclusters associated with $M$.

Clearly, biclustering is a highly combinatorial problem with a search space size $O(2^{|I|+|J|})$. In its general case, biclustering is Nondeterministic Polynomial (NP)-hard [9, 22].

## 29.2 TYPES OF BICLUSTERS

A bicluster can occur in one of the following cases:

1. A bicluster with constant values is a bicluster in which all the values are equal to a constant $c$:

$$m_{ij} = c \tag{29.2}$$

2. Bicluster with constant values on rows or columns:
   - A bicluster with constant values on rows is a bicluster in which all the values can be obtained by using one of the following equations:

$$m_{ij} = c + a_i \tag{29.3}$$

$$m_{ij} = ca_i \tag{29.4}$$

   where $c$ is a constant and $a_i$ is the adjustment for the row $i$, $1 \leq i \leq n$.

- A bicluster with constant values on columns is a bicluster in which all the values can be obtained by using one of the following equations:

$$m_{ij} = c + b_j \tag{29.5}$$

$$m_{ij} = cb_j \tag{29.6}$$

  where $c$ is a constant and $b_j$ is the adjustment for the column $j$, $1 \leq j \leq m$.

3. A bicluster with coherent values is a bicluster that can be obtained by using one of the following equations:

$$m_{ij} = c + a_i + b_j \tag{29.7}$$

$$m_{ij} = ca_i b_j \tag{29.8}$$

4. A bicluster with linear coherent values is a bicluster in which all the values can be obtained by using the following equation:

$$m_{ij} = ca_i + b_j \tag{29.9}$$

5. A bicluster with a coherent evolution is a bicluster in which all the rows (respectively columns) induce a linear order across a subset of columns (respectively rows).

## 29.3 GROUPS OF BICLUSTERS

A group of biclusters can occur in one of the following cases [22]:

1. A single bicluster (Figure 29.1(a))
2. An exclusive rows and columns group of biclusters (Figure 29.1(b))
3. A nonoverlapping group of biclusters with a checkerboard structure (Figure 29.1(c))
4. An exclusive rows group of biclusters (Figure 29.1(d))
5. An exclusive columns group of biclusters (Figure 29.1(e))
6. A nonoverlapping group of biclusters with a tree structure (Figure 29.1(f))
7. A nonoverlapping nonexclusive group of biclusters (Figure 29.1(g))
8. An overlapping group of biclusters with a hierarchical structure (Figure 29.1(h))
9. An arbitrarily positioned overlapping group of biclusters (Figure 29.1(i))

A natural way to visualize a group of biclusters consists in assigning a different color to each bicluster and of reordering the rows and the columns of the data matrix so that we obtain a data matrix with colored blocks in which each block represents a bicluster.

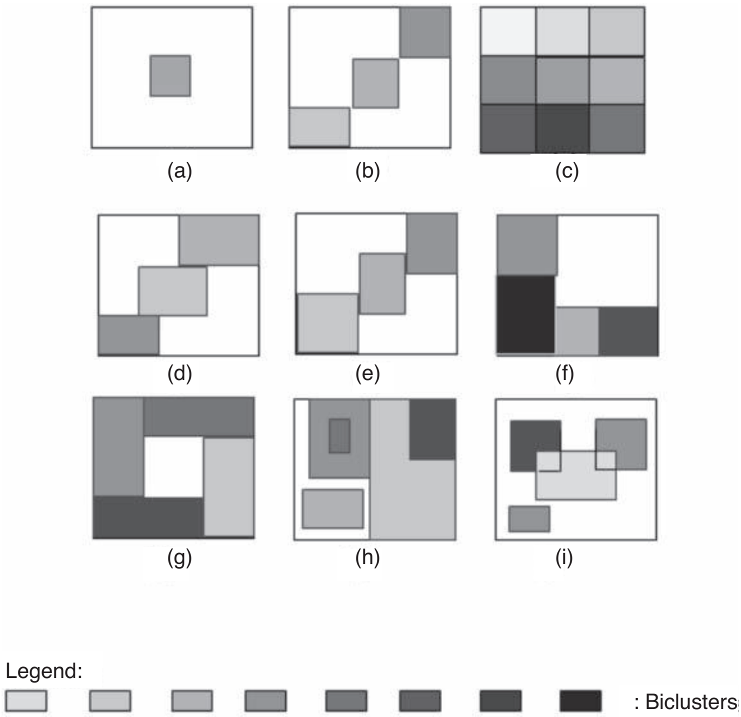


**Figure 29.1** Possible structures of a group of biclusters in a data matrix.

## 29.4 EVALUATION FUNCTIONS

An *evaluation function* is an indicator of the coherence degree of a bicluster in a data matrix. There are several evaluation functions of a bicluster [2, 3, 8, 9, 13, 17, 34]. The most popular evaluation function is the *Mean Squared Residue* (MSR) function ($E_{\mathrm{MSR}}$) proposed in [9] (Equation 29.10). It is used by several algorithms like [2, 6, 8, 12, 24, 37, 39]. $E_{\mathrm{MSR}}$ represents the variance of a particular subset of rows under a particular subset of columns with respect to the coherence.

$$E_{\mathrm{MSR}}(I', J') = \frac{\sum_{i\in I', j\in J'}(m_{ij} - m_{iJ'} - m_{I'j} + m_{I'J'})^2}{|I'||J'|} \tag{29.10}$$

where $m_{I'J'}$ is the average over the whole bicluster, $m_{I'j}$ is the average over the column $j$, and $m_{iJ'}$ is the average over the row $i$.

A low (respectively high) $E_{\mathrm{MSR}}$ value (*i.e.*, *close* to 0 [respectively higher than a fixed threshold]) indicates that the bicluster is strongly (respectively weakly) coherent. If a bicluster has a value of $E_{\mathrm{MSR}}$ lower than a given threshold $\delta$, then it is called $\delta$*−bicluster*. However, the $E_{\mathrm{MSR}}$ function is deficient to assess certain types of biclusters [1, 28, 34].

Angiulli *et al.* [2] and, Divina and Aguilar-Ruiz [13] propose using $E_{\text{MSR}}$, the row variance $E_{\text{RV}}$, and the volume $E_{\text{V}}$ of a bicluster. The row variance $E_{\text{RV}}$ is defined as follows:

$$E_{\text{RV}}(I', J') = \frac{\sum_{i \in I', j \in J'} (m_{ij} - m_{iJ'})^2}{|I'||J'|} \tag{29.11}$$

Biclusters that contain genes with large changes in their expression values under different conditions are characterized by high values of the row variance. It follows that the row variance can be used to guarantee that a bicluster captures genes exhibiting coherent trends under some subset of conditions.

The volume $E_{\text{V}}(I', J')$ of a bicluster $(I', J')$ is used to maximize the size of this bicluster. It is defined by:

$$E_{\text{V}}(I', J') = |I'||J'| \tag{29.12}$$

Das *et al.* [11] propose finding the maximum-sized bicluster that does not exceed a certain coherence constraint. The coherence is expressed as an MSR score. Hence, Das *et al.* try to maximize the volume $E_{\text{V}}(I', J')$ (Equation 29.12) and to find biclusters with a value of $E_{\text{MSR}}$ lower than a given threshold $\delta$, $\delta \geq 0$ (Equation 29.10).

Teng and Chan [34] propose the *Average Correlation Value* (ACV) function, $E_{\text{ACV}}$. It is defined by the following equation:

$$E_{\text{ACV}}(I', J') = \max \left\{ \frac{\sum_{i \in I'} \sum_{j \in I'} r_{ij} - |I'|}{|I'|(|I'|-1)}, \frac{\sum_{k \in J'} \sum_{l \in J'} r_{kl} - |J'|}{|J'|(|J'|-1)} \right\} \tag{29.13}$$

where $r_{ij}$ $(i \neq j)$ is the Pearson's correlation coefficient associated with the row indices $i$ and $j$ in the bicluster $(I', J')$ [26], and $r_{kl}$ $(k \neq l)$ is the Pearson's correlation coefficient associated with the column indices $k$ and $l$ in the bicluster $(I', J')$.

A high (respectively low) $E_{\text{ACV}}$ value (*i.e.*, *close* to 1 [respectively *close* to 0]) indicates that the bicluster is strongly (respectively weakly) coherent. However, the performance of the $E_{\text{ACV}}$ function decreases when noise exists in the data matrix [8].

Cheng *et al.* [8] propose using the $E_{\text{ACV}}$ and $E_{\text{MSR}}$ functions. A bicluster with a high coherence has a low $E_{\text{MSR}}$ value and a high $E_{\text{ACV}}$ value.

Ayadi *et al.* [3] propose the *Average Spearman's Rho* (ASR) function, $E_{\text{ASR}}$. It is defined by the following equation:

$$E_{\text{ASR}}(I', J') = 2 \max \left\{ \frac{\sum_{i \in I'} \sum_{j \in I', j \geq i+1} \rho_{ij}}{|I'|(|I'|-1)}, \frac{\sum_{k \in J'} \sum_{l \in J', l \geq k+1} \rho_{kl}}{|J'|(|J'|-1)} \right\} \tag{29.14}$$

where $\rho_{ij}$ $(i \neq j)$ is the Spearman's rank correlation associated with the row indices $i$ and $j$ in the bicluster $(I', J')$ [19], and $\rho_{kl}$ $(k \neq l)$ is the Spearman's rank correlation associated with the column indices $k$ and $l$ in the bicluster $(I', J')$.

Let us note that the values of the Spearman's rank correlation belong to $[-1..1]$. A high (respectively low) Spearman's rank correlation value (*i.e.*, *close* to 1 [respectively *close* to –1]) indicates that the two vectors are strongly (respectively weakly) coherent [19]. So, the values of the $E_{\mathrm{ASR}}$ function belong also to [–1..1]. Hence, a high (respectively low) $E_{\mathrm{ASR}}$ value (*i.e.*, *close* to 1 [respectively *close* to –1]) indicates that the bicluster is strongly (respectively weakly) coherent. Furthermore, it has been shown that Spearman's rank correlation is less sensitive to the presence of noise in data. Since the $E_{\mathrm{ASR}}$ function is based on Spearman's rank correlation, $E_{\mathrm{ASR}}$ is also less sensitive to the presence of noise in data.

## 29.5 SYSTEMATIC AND STOCHASTIC BICLUSTERING ALGORITHMS

As we mentioned in the introduction of this chapter, the biclustering problem is NP-hard [9, 22]. There are several heuristic algorithms to construct a group of biclusters *close* to the optimal one.

We distinguish two main classes of biclustering algorithms: *systematic search algorithms* and *stochastic search* algorithms, also called *metaheuristic algorithms*.

By using a *systematic search algorithm*, we adopt one of the following approaches:

1. *Divide-And-Conquer* (DAC) approach: By adopting this approach, we start with a bicluster representing the whole data matrix then we partition this matrix in two submatrices to obtain two biclusters. We reiterate recursively this process until we obtain a certain number of biclusters verifying a certain number of properties. The advantage of this approach is that it is fast; however, its biggest disadvantage is that it may ignore good biclusters by partitioning them before identifying them. Representative examples of algorithms adopting this approach are given by Dufiy and Quiroz [14], Hartigan [17] and Prelic *et al.* [29].
2. *Greedy Iterative Search* (GIS) approach: By adopting this approach, at each iteration, we construct submatrices of the data matrix by adding/removing a row/column to/from the current submatrix that maximizes/minimizes a certain function. We reiterate this process until no other row/column can be added/removed to/from any submatrix. This approach presents the same advantage and the same disadvantage of the previous approach. Representative examples of algorithms adopting this approach are given by Ben-Dor *et al.* [5], Cheng *et al.* [8], Cheng and Church [9], Liu and Wang [21], Teng and Chan [34], and Yang *et al.* [37, 38].
3. *Biclusters Enumeration* (BE) approach: By adopting this approach, we identify all the possible groups of biclusters to keep the *best* one. It is clear that the

advantage of this approach is that it enables obtaining the best solution, and its disadvantage is that it is costly in computing time and in memory space. Representative examples of algorithms adopting this approach are given by Ayadi *et al.* [3], Liu and Wang [20], Okada *et al.* [27] and Tanay *et al.* [32].

By using a *stochastic search algorithm*, we adopt one of the following approaches:

1. *Neighborhood Search* (NS) approach: By adopting this approach, we start with an initial solution that can be either a cluster, a bicluster, or the whole matrix. Then, at each iteration, we try to improve this solution by adding and/or removing some neighbor genes/conditions to minimize/maximize a certain function. The difference with the systematic greedy search algorithms is that if we delete, for example, one neighbor gene/condition, then we can later add this neighbor to the solution if it improves this solution. The advantage of this approach lies in the ability to control the running time. Indeed, when the quality of a solution tends to improve gradually over time, the user can stop the execution at a chosen time. The disadvantage of this approach is that the search may stop even if the best solution found by the algorithm is not optimal. Representative examples of algorithms adopting this approach are given by Bryan *et al.* [7], and Dharan and Nair [12].
2. *Evolutionary Computation* (EC) approach: By adopting this approach, we try to use the principle of the nature evolutionary process. By using such algorithms, we start from an initial population of solutions (*i.e.*, clusters, biclusters, or the whole matrix), and then, we measure the quality of each solution of the population by an evaluation mechanism. We select several solutions to produce new solutions by recombination and mutation operators. This process ends when the stop condition is verified. The main advantage of the algorithms adopting this approach is that they are less apt to get trapped in local optima compared with the algorithms that proceed from point to point in the solution space. However, their main disadvantage is their inability to guarantee global optimality. Actually, these algorithms do not always recognize the optimum even after they find it. A representative example of algorithms adopting this approach is given by Divina and Aguilar-Ruiz [13].
3. *Hybrid* (H) approach: By adopting this approach, we try to combine both the neighborhood search approach and the evolutionary approach. Indeed, this approach enables us to benefit from the power of both the neighborhood search approach and the evolutionary approach. Representative examples of algorithms adopting this approach are given by Bleuler *et al.* [6], Gallo *et al.* [15] and Mitra and Banka [24].

Table 29.2 is a synoptic table of biclustering algorithms, where

- $n$ is the number of genes
- $m$ is the number of conditions

**Table 29.2 Synoptic table of biclustering algorithms**

| Reference | Approach | Time |
|---|---|---|
| Prelic *et al.* [29] | DAC | $O(nm\beta \min\{n, m\})$ |
| Angiulli *et al.* [2] | GIS | $O(\mu C_u[(1 - p_r)(n + m) + p_r])$ |
| Ben-Dor *et al.* [5] | GIS | $O(nm^3 l)$ |
| Cheng and Church [9] | GIS | $O(mn)$ |
| Cheng *et al.* [8] | GIS | $O(n^2 m^4 (n\log(n) + m\log(m)))$ |
| Liu and Wang [21] | GIS | $O(nm(n + m)^2)$ |
| Teng and Chan [34] | GIS | |
| Ayadi *et al.* [3] | BE | $O(2^n m^2 n^2)$ |
| Tanay *et al.* [32] | BE | $O((n2^{d+n})^{\log \frac{r+1}{r}(rd)})$ |
| Wang *et al.* [35] | BE | |
| Bryan *et al.* [7] | NS | |
| Dharan and Nair [12] | NS | $O(mn(m + n))$ |
| Divina and Aguilar-Ruiz [13] | EC | |
| Bleuler *et al.* [6] | H | |
| Gallo *et al.* [15] | H | |
| Mitra and Banka [24] | H | |

- $l$ is the number of the best partial models of order
- $C_u$ is the cost of computing the new residue and the new row variance of the bicluster after performing a move
- $p_r$ is a user-provided probability that the algorithm is allowed to execute a random move
- $d$ is the bounded degree of gene vertices in a bipartite graph $G$ whose two sides correspond to the set of genes and the set of conditions
- $r$ is the maximum weight edge in the bipartite graph $G$
- $\beta$ is the number of biclusters that are not contained entirely in any other bicluster
- $\mu$ is the maximum number of times that a flip can be done

We now present briefly some biclustering tools that are publicly available for microarray data analysis.

1. GEMS [36] is a web server for biclustering of microarray data. It is based on a *Gibbs* sampling paradigm [31]. GEMS is available at http://genomics10.bu.edu/terrence/gems/.
2. BICAT [4] is a biclustering analysis toolbox that is used mostly by the community and contains several implementations of biclustering algorithms like the *Order Preserving SubMatrix* (OPSM) algorithm [5], Cheng and Church's algorithm [9], the *Iterative Signature Algorithm* (ISA) [18], the

*XMOTIF* algorithm [25] and the *BIMAX* algorithm [29]. BICAT is available at http://www.tik.ee.ethz.ch/sop/bicat.

3. *BIVISU* [8] is a biclustering algorithm based on *Parallel Coordinate* (PC) formulation. It can visualize the detected biclusters in a two-dimensional setting by using PC plots. *BIVISU* is available at http://www.eie.polyu.edu.hk/nflaw/Biclustering/index.html.
4. *Bayesian BiClustering* model (BBC) [16] is a biclustering algorithm based on the Monte Carlo procedure. BBC is available at http://www.people.fas.harvard.edu/junliu/BBC/.
5. *BICOVERLAPPER* [30] is a visual framework that supports:
   - Simultaneous visualization of one or more sets of biclusters
   - Visualization of microarray data matrices as heatmaps and PC
   - Visualization of *transcription regulatory networks*
   - Linkage of different visualizations and data to achieve a broader analysis of an experiment results

   *BICOVERLAPPER* is available at http://vis.usal.es/bicoverlapper/.
6. *E-CCC-BICLUSTERING* [23] is a biclustering algorithm that can find and report all the maximal contiguous column coherent biclusters with approximate expression patterns. *E-CCC-BICLUSTERING* is available at http://kdbio.inesc-id.pt/software/e-ccc-biclustering.

## 29.6 BIOLOGICAL VALIDATION

Biological validation can evaluate qualitatively the capacity of an algorithm to extract meaningful biclusters from a biological point of view. Assessing the biological meaning of the results of a biclustering algorithm is not a trivial task because general guidelines do not exist in the literature on how to achieve this task. Several authors use artificial datasets to validate their approaches. However, an artificial scenario is inevitably biased regarding the underlying model and only reflects certain aspects of biological reality. To assess statistically a group of genes biclusters, we can use *Gene Ontology* (GO) annotation [10]. Unfortunately, GO annotation enables validating just subsets of genes but not subsets of genes under subsets of conditions.

In GO, genes are assigned to three structured, controlled vocabularies (*i.e.*, *ontologies*, that describe genes products in terms of associated *biological processes, components*, and *molecular functions* in a species-independent manner). Users measure the degree of enrichment (*i.e.*, $p$-values) by using a cumulative *hypergeometric* distribution that involves the probability of observing the number of genes from a particular GO category (*i.e.*, *biological processes*, *components*, and *molecular functions*) within each bicluster. Statistical significance is evaluated for the genes in each bicluster by computing $p$-values, which indicate how well they match with the different GO categories. Let us note that a smaller $p$-value, *close* to 0, is indicative of a better match [33].

The *Gene Ontology Consortium* (GOC) [10] (http://www.geneontology.org) is involved in the development and application of the GO. We present examples of web-tools related to GOC as follows:

1. *GO Term Finder* (http://db.yeastgenome.org/cgi-bin/GO/goTermFinder) searches for significant shared GO terms, or parents of GO terms, used to annotate genes products in a given list.
2. *FuncAssociate* (http://llama.med.harvard.edu/cgi/func/funcassociate) is a web-based tool that accepts as input a list of genes and returns a list of GO attributes that are over-underrepresented among the genes of the input list. Only those over-underrepresented genes are reported.
3. *GENECODIS* (http://genecodis.dacya.ucm.es/) is a web-based tool for the functional analysis of a list of genes. It integrates different sources of information to search for annotations that frequently cooccur in a list of genes and ranks them according to their statistical significance.

The microarray datasets presented in Table 29.3 are such that each experimental condition corresponds to a patient presenting a kind of pathology. For example, the *Leukemia* dataset discriminates patients affected by either *Lymphoblastic* or *Myeloid leukemia*. Thus, we do not know the biological coherence between genes, although we know the medical classification of conditions. In this case, we can evaluate the ability of an algorithm to separate the samples according to their known classification. To this end, we can compute the number of columns labeled with the same class and belonging to the same bicluster. Obviously, the higher the number of columns in

**Table 29.3 Microarray datasets used to evaluate biclustering algorithms**

| Dataset | Nbr. Genes | Nbr. Conditions | Website |
|---|---|---|---|
| Arabidopsis thaliana | 734 | 69 | http://www.tik.ethz.ch/sop/bimax/ |
| Colon rectal cancer | 2000 | 62 | http://microarray.princeton.edu/oncology/affydata/index.html |
| Colon tumor | 2000 | 62 | http://sdmc.lit.org.sg/GEDatasets/Datasets.html |
| Human lymphoma | 4026 | 96 | http://arep.med.harvard.edu/biclustering/ |
| Leukemia | 7129 | 72 | http://sdmc.lit.org.sg/GEDatasets/Datasets.html |
| Lung cancer | 12,533 | 181 | http://sdmc.lit.org.sg/GEDatasets/Datasets.html |
| Ovarian cancer tumor | 15,154 | 253 | http://sdmc.lit.org.sg/GEDatasets/Datasets.html |
| Prostate cancer | 12,600 | 136 | http://sdmc.lit.org.sg/GEDatasets/Datasets.html |
| Saccharomyces cerevisiae | 2993 | 173 | http://www.tik.ethz.ch/sop/bimax/ |
| Yeast cell cycle | 2884 | 17 | http://arep.med.harvard.edu/biclustering/ |

a bicluster labeled with the same class label, the higher its biological quality. In fact, this means that many patients with the same diagnosis are grouped together with respect to a subset of genes; thus, we could induce that those genes probably have a similar functional category and characterize the majority class of patients.

## 29.7 CONCLUSION

In this chapter, we have reviewed biclustering algorithms of microarray data. Advantages and disadvantages of these algorithms are reported.

Although biclustering of microarray data has been the subject of a large research, noone of the existing biclustering algorithms are perfect, and the construction of biologically significant groups of biclusters for large microarray data is still a problem that requires a continuous work.

Biological validation of biclustering algorithms of microarray data is one of the most important open issues. So far, there are no general guidelines in the literature on how to validate biologically a biclustering algorithm.

## REFERENCES

1. J.S. Aguilar-Ruiz. Shifting and scaling patterns from gene expression data. *Bioinformatics*, 21:3840–3845, 2005.
2. F. Angiulli, E. Cesario, and C. Pizzuti. Random walk biclustering for microarray data. *J Inform Sci*: 1475–1497, 2008.
3. W. Ayadi, M. Elloumi, and J.K. Hao. A biclustering algorithm based on a bicluster enumeration tree: Application to DNA microarray data, 2:2009.
4. S. Barkow, S. Bleuler, A. Prelic, P. Zimmermann, and E. Zitzler. Bicat: A biclustering analysis toolbox. *Bioinformatics*, 22(10):1282–1283, 2006.
5. A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini. Discovering local structure in gene expression data: The order-preserving submatrix problem. *RECOMB '02: Proceedings of the Sixth Annual International Conference on Computational Biology*. ACM, New York: 49–57, 2002.
6. S. Bleuler, A. Prelic, and E. Zitzler. An ea framework for biclustering of gene expression data. *Proceedings of Congress on Evolutionary Computation*: 166–173, 2004.
7. K. Bryan, P. Cunningham, and N. Bolshakova. Application of simulated annealing to the biclustering of gene expression data. *IEEE Trans Inform Tech Biomed*, 10(3):519–525, 2006.
8. K.O. Cheng, N.F. Law, W.C. Siu, and A.W. Liew. Identification of coherent patterns in gene expression data using an efficient biclustering algorithm and parallel coordinate visualization. *BMC Bioinformatics*, 9(210):1282–1283, 2008.
9. Y. Cheng and G.M. Church. Biclustering of expression data. *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Menlo Park, CA: 93–103, 2000.
10. Gene Ontology Consortium. Gene ontology: Tool for the unification of biology. *Nat Genet*, 25:25–29, 2000.

11. R. Das, S. Mitra, H. Banka, and S. Mukhopadhyay. Evolutionary biclustering with correlation for gene interaction networks. *PReMI, LNCS*: 416–424, 2007.
12. A. Dharan and A.S. Nair. Biclustering of gene expression data using reactive greedy randomized adaptive search procedure. *BMC Bioinformatics*, 10(Suppl 1):S27, 2009.
13. F. Divina and J.S. Aguilar-Ruiz. A multi-objective approach to discover biclusters in microarray data. *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York: 385–392, 2007.
14. D. Dufiy and A. Quiroz. A permutation based algorithm for block clustering. *J Classif*, (8):65–91, 1991.
15. C.A. Gallo, J.A. Carballido, and I. Ponzoni. Microarray biclustering: A novel memetic approach based on the pisa platform. *EvoBIO '09: Proceedings of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Springer-Verlag, Berlin, Germany: 44–55, 2009.
16. J. Gu and J.S. Liu. Bayesian biclustering of gene expression data. *BMC Genom*, 9(Suppl 1):S4, 2008.
17. J.A. Hartigan. Direct clustering of a data matrix. *J Am Stat Assoc*, 67(337):123–129, 1972.
18. J. Ihmels, S. Bergmann, and N. Barkai. Defining transcription modules using large-scale gene expression data. *Bioinformatics*, 13:1993–2003, 2004.
19. E.L. Lehmann and H.J.M. D'Abrera. Nonparametrics: Statistical Methods Based on Ranks, *revised edition*. Englewood Cliffs, NJ, Prentice-Hall: 292–323, 1998.
20. J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space. *IEEE Int Conf Data Mining*: 187–194, 2003.
21. X. Liu and L. Wang. Computing the maximum similarity bi-clusters of gene expression data. *Bioinformatics*, 23(1):50–56, 2007.
22. S.C. Madeira and A.L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Trans Computat Biol Bioinform*, 1(1):24–45, 2004.
23. S.C. Madeira and A.L. Oliveira. A polynomial time biclustering algorithm for finding approximate expression patterns in gene expression time series. *Algorithm Mol Biol*, 4: 2009.
24. S. Mitra and H. Banka. Multi-objective evolutionary biclustering of gene expression data. *Pattern Recogn*, 39(12):2464–2477, 2006.
25. T.M. Murali and S. Kasif. Extracting conserved gene expression motifs from gene expression data. *Pac Symp Biocomput*, 8:77–88, 2003.
26. J.L. Myers and D.W. Arnold. *Research Design and Statistical Analysis*. Routledge, New York, 2003.
27. Y. Okada, K. Okubo, P. Horton, and W. Fujibuchi. Exhaustive search method of gene expression modules and its application to human tissue data. In *IAENG Int J Comput Sci*, 34:1–16, 2007.
28. B. Pontes, F. Divina, R. Giráldez, and J.S. Aguilar-Ruiz. Virtual error: A new measure for evolutionary biclustering. In E. Marchiori, J.H. Moore, and J.C. Rajapakse, editors, *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 4447 of *Lecture Notes in Computer Science*. Springer, New York: 217–226, 2007.
29. A. Prelic, S. Bleuler, P. Zimmermann, P. Buhlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22(9):1122–1129, 2006.

30. R. Thern, R. Santamara, and L. Quintales. A visual analytics approach for understanding biclustering results from microarray data. *BMC Bioinformatics*, 9:2008.

31. Q. Sheng, Y. Moreau, and B. De Moor. Biclustering microarray data by gibbs sampling. *Bioinformatics*, 19:II196–II205, 2003.

32. A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18:S136–S144, 2002.

33. S. Tavazoie, J.D. Hughes, M.J. Campbell, R.J. Cho, and G.M. Church. Systematic determination of genetic network architecture. *Nat Genet*, 22:281–285, 1999.

34. L. Teng and L. Chan. Discovering biclusters by iteratively sorting with weighted correlation coefficient in gene expression data. *J Signal Process Syst*, 50(3):267–280, 2008.

35. H. Wang, W. Wang, J. Yang, and P.S. Yu. Clustering by pattern similarity in large data sets. *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY: 394–405, 2002.

36. C.J. Wu and S. Kasif. Gems: A web server for biclustering analysis of expression data. *Nucleic Acids Res*, 33:W596–W599, 2005.

37. J. Yang, H. Wang, W. Wang, and P. Yu. Enhanced biclustering on expression data. *BIBE '03: Proceedings of the 3rd IEEE Symposium on BioInformatics and BioEngineering*. IEEE Computer Society, Washington, DC: 321–327, 2003.

38. Y.H. Yang, S. Dudoit, P. Luu, D.M. Lin, V. Peng, J. Ngai, and T.P. Speed. Normalization for cdna microarray data: A robust composite method addressing single and multiple slide systematic variation. *Nucleic Acids Res*, 30:1–12, 2002.

39. Z. Zhang, A. Teo, B.C. Ooi, and K.L. Tan. Mining deterministic biclusters in gene expression data. *IEEE International Symposium on Bioinformatic and Bioengineering*: 283–290, 2004.

# 30

# COMPUTATIONAL MODELS FOR CONDITION-SPECIFIC GENE AND PATHWAY INFERENCE

Yu-Qing Qiu, Shihua Zhang, Xiang-Sun Zhang, and Luonan Chen

## 30.1 INTRODUCTION

High-throughput experimental data such as protein–protein interaction [49, 57], gene expression [11], and ChIP-chip data [48], are now explored widely to study the complicated behaviors of living organisms from various aspects at a molecular level. For example, DNA microarrays provide us with a key step toward the goal of uncovering gene function on a global scale and biomolecular networks pave the way for understanding the whole biological systems on a systematic level [28, 69]. These studies are very important because proteins do not function in isolation but rather interact with one another and with various molecules (*e.g.*, DNA, RNA, and small molecules) to form molecular machines (pathways, complexes, or functional modules). These machines shape the modular organization of biological systems, represent static and dynamic biological processes, and transmit/respond to intra- and extracellular signals. In other words, modules are basic functional building block of biological systems that have been observed in various types of network data including protein–protein interaction networks, metabolic networks, transcriptional regulation networks, and gene coexpression networks [63]. In contrast to individual components, it has been recognized that modules as well as biomolecular

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

networks are ultimately responsible to the forms and functions of living organisms, and also can also reasonably explain the causes of various phenotypes.

In real biological systems, although some genes (*e.g.*, house-keeping genes), are expressed constitutively under various conditions and tissues to carry out basic cellular process for growth and sustenance, most genes or pathways are actually active only under defined conditions (*e.g.*, specific time and tissue). These condition-specific modules are important to uncover the molecular functional mechanism in different development stages. Moreover, their identification can give insights into understanding human diseases.

Although protein interaction networks or pathways are available for many organisms based on accumulated protein interaction data, it is still a difficult task to identify condition-specific genes, pathways, or modules corresponding to the changing conditions and environments in living cells. To decipher these biological mechanisms, a single type of experimental data only provides limited information. For example, protein-protein interaction data only tell of possible interactions among proteins rather than when and where they interact. On the other hand, gene expression experimental data could profile genes' dynamic functional behavior. Therefore, combining network data with microarray data provides a powerful way to study specific expression patterns, active modules, or even disease-related pathways under different conditions. Additionally, phenotype similarity networks derived from literature mining and other sources of data are also useful for phenotype specific gene or disease gene prediction, as well as module network construction.

How to integrate these heterogeneous data to elucidate biological mechanisms is an essential and challenging problem in computational biology and systems biology. In this chapter, we surveyed computational approaches for identifying condition-specific genes, pathways, or modules. Two kinds of approaches are introduced in detail. One is to combine gene expression data with prior-known pathway information, and the other is to integrate gene expression data and large-scale biological interaction networks. We will extend the survey for highly related topics including genetic-related pathways identification and typical disease gene and pathway prediction (prioritization) by integrating various experiments and text mining information. To interpret the relationship of modules in a high level, the concept of a modular network in which nodes are modules also was introduced briefly.

## 30.2 CONDITION-SPECIFIC PATHWAY IDENTIFICATION

The expression changes of genes reflect their activities in the transcriptional level. In different biological processes or conditions such as disease or nondisease, condition-specific genes and pathways are likely to be expressed differentially. By using this important cue, many computational methods for identifying condition-specific pathways have been proposed. The underlying methodologies can be classified into two classes (see Figure 30.1). By integrating known gene sets (derived from pathway database, groups of genes with similar functions, *etc.*), gene set analysis methods use statistical methods to detect the sets of genes that have significant expression changes

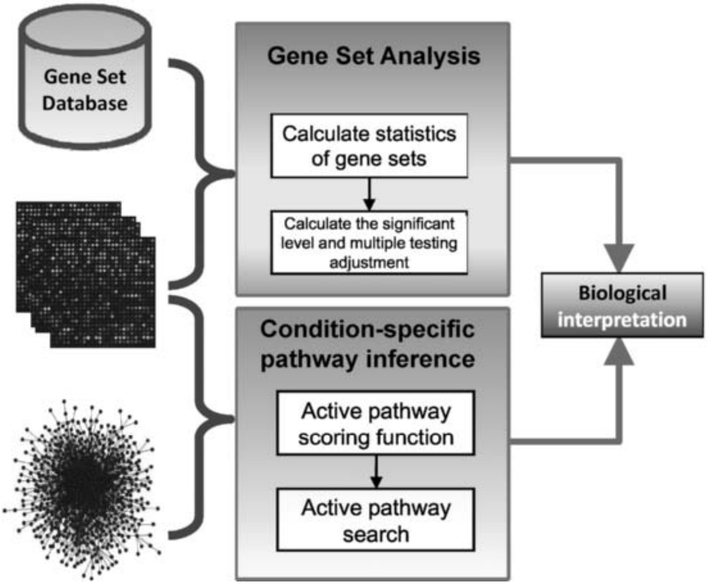

Gene Set Database
Gene Set Analysis
Calculate statistics of gene sets
Calculate the significant level and multiple testing adjustment
Condition-specific pathway inference
Active pathway scoring function
Active pathway search
Biological interpretation

**Figure 30.1** Pipeline of condition-specific pathway identification methods.

in different conditions. The consistent expression pattern of groups of genes that may be ignored by analyzing genes individually can be captured by these methods. However, because literature confirmed that pathways are far from covering all cellular molecular mechanisms, gene set analysis methods cannot discover new pathways. Thus, other types of condition-specific pathway inference methods are presented by integrating the biomolecular interaction networks and gene expression data to discover subnetworks that show significant expression changes or active behaviors. These subnetworks can be biological complexes, signaling pathways, or group of genes with similar functions, and so on, and the within interactions can explain the cooperation between member genes. Finally, the identified condition-specific pathways or modules can be used for making biological conclusions and proposing new biological hypotheses. We will survey the highly related methods in the following sections.

### 30.2.1 Gene Set Analysis

Microarray data are often used to identify differentially expressed genes. Many methods such as $t$-test and SAM (significance analysis of microarrays) [61] have been developed to evaluate the expression change level of each gene between two groups of samples. These methods usually choose genes which shows significant expression changes than an arbitrary selected threshold to form a gene list for further biological interpretation, such as Gene Ontology (GO) terms or pathways enrichment analysis.

A major drawback of this kind of method is that the choice of threshold greatly affects the biologic conclusions [44]. Moreover, the subtle but coherent expression changes that often occur in a pathway are discarded [7, 10]. And these methods all are based on the assumption of independence among genes which increases false positive prediction.

To overcome these problems, gene set analysis methods were proposed [40]. Rather than scoring a single gene, such methods evaluate the differential expression level of predefined gene sets such as pathways, GO categories, or functional modules detected by computational methods. As illustrated in [38], a set of genes with moderate but coordinated expression changes was identified by the gene set analysis method but failed to be recognized by the individual gene analysis methods because genes function in concert rather than in isolation. Functionally related genes often display a coordinated expression profiles to achieve a specific function in a cell. Thus, even weak expression changes in individual genes gathered to a large gene set can show a significant macrochange in gene set level.

The principle of gene set analysis methods are illustrated in Figure 30.1. The functionally related gene set is collected from databases such as GO [14], KEGG [30], and GenMAPP [15]. Using the preprocessed gene expression data, one designed statistic is evaluated for each gene set, which measures the expression deviation from the null hypothesis of no association with the phenotype. The statistical significance (or $p$-value) for each gene set is evaluated by permuting sample or gene labels because analytical distribution of the statistic is often difficult to obtain. Meanwhile the multiple testing adjustment has to be used to correct $p$-values of each gene set to reduce the false positive rate. Finally, the gene sets with high significance are reported as biologically relevant pathways for further biological interpretation.

The gene set analysis methods are inspired by the work of Mootha *et al.* [38], who proposed a method named gene set enrichment analysis (GSEA). They applied GSEA to a gene expression dataset of human normal and diabetes tissues and successfully identified a set of genes involved in oxidative phosphorylation whose expression coordinately altered in human diabetic muscle. The changes are subtle at the level of individual genes so that they are ignored by individual gene analysis method. The approach of GSEA ranks genes based on the signal-to-noise ratio and then uses the Kolmogorove–Smirnov statistic as a scoring function to represent the relative enrichment of differentially expressed genes in each gene set. The significance of the entire dataset is calculated in two ways after the sample permutation. The first is to count the number of sample permutations for which a gene set with a better score than the observed is found. The second is to count the number of gene sets that have a better score than a given threshold for the real dataset and permuted datasets. The first one assigns $p$-values to each gene set, whereas the second one assigns $p$-values only to the entire dataset. Because of its high performance and user-friendly desktop application and website, GSEA is currently the most popular method for gene set analysis. Later, Subramanian *et al.* [58] made a direct improvement to the Kolmogorov–Smirnov statistic via weighting each gene using its correlation with phenotypes to prevent gene sets clustered in the middle of the ordered gene list from getting high scores.

**Table 30.1 Tools for gene set analysis**

| Name | Statistical Methods | URL |
|---|---|---|
| ADGO | $z$-score | http://array.kobic.re.kr/ADGO |
| ALLEZ | $z$-score | http://www.stat.wisc.edu/newton/ |
| ASSESS | Sample randomization | http://people.genome.duke.edu/jhg9/assess/ |
| CATMAP | Gene randomization | http://bioinfo.thep.lu.se/catmap.html |
| ERMINEJ | Wilcoxon rank-sum test | http://www.bioinformatics.ubc.ca/ermineJ/ |
| EU.GENE. .ANALYZER | Fisher's exact test | http://www.ducciocavalieri.org/bio/Eugene.htm |
| FATISCAN | Fisher's exact test; Hypergeometric test | http://fatiscan.bioinfo.cipf.es/ |
| FUNCLUSTER | Fisher's exact | http://corneliu.henegar.info/FunCluster.htm |
| GAZER | $z$-score; permutation | http://integromics.kobic.re.kr/GAzer/index.faces |
| GENETRAIL | Fisher's exact test; hypergeometric test; sample randomization; | http://genetrail.bioinf.uni-sb.de/ |
| GLOBALTEST | Sample randomization | http://bioconductor.org/packages/2.0/bioc/html/globaltest.html |
| GOAL | Gene randomization | http://microarrays.unife.it |
| GO-MAPPER | Gaussian distribution; EQ-score | http://www.gatcplatform.nl/gomapper/index.php |
| GOTM | Hypergeometric | http://www.gotm.net/ |
| GSA | Sample randomization | http://www-stat.stanford.edu/ tibs/GSA/ |
| GSEA | Sample randomization; Kolmogorov-Smirnov test | http://www.broad.mit.edu/gsea/ |
| JPROGO | Fisher's exact test; Kolmogorov-Smirnov test; $t$-test; unpaired Wilcoxon's test | http://www.jprogo.de/ |
| MAPPFINDER | $z$-score | http://www.genmapp.org |
| MEGO | $z$-score | http://www.dxy.cn/mego/ |
| PAGE | $z$-score | From the author (kimsy@kribb.re.kr) |
| PLAGE | Sample randomization | http://dulci.biostat.duke.edu/pathways/ |
| SAFE | Sample randomization | http://bioconductor.org/packages/2.0/bioc/html/safe.html |
| SAM-GS | Sample randomization | http://www.ualberta.ca/Byyasui/homepage.html |
| T-PROFILER | $t$-test | http://www.t-profiler.org/ |

Besides GSEA, various gene set analysis methods have been proposed (Table 30.1) in recent years. MAPPFINDER [20] employs an approach based on differentially expressed gene hit-counting and the hypergeometric distribution to score gene sets. Permutations and multiple testing corrections are used to provide statistically significant gene sets. Efron and Tibshirani [21] introduced five statistics and tested on five simulated examples. They found that the *maxmean* statistic is the only method with consistently low $p$-values in all situations. Barry *et al.* [5] proposed significance analysis function and expression (SAFE), which extends GSEA to cover multiclass, continuous, and survival phenotypes and couples with more statistic options including Wilcoxon rank sum, Kolmogorov–Smirnov, and hypergiometric statistic. GLOBALTEST [24] proposed by Goeman *et al.* employs a scoring test based on a random-effect logistic model fitted for each gene set and subsequently extends it to multiclass, continuous, and survival phenotypes and allows for covariate adjustments. FATISCAN [1] uses the segmentation test approach, which consists of the sequential applications of a Fisher's exact test over the contingency tables formed with ordered gene lists. This method does not depend on the original data from which the ranking of the list is derived and can be applied to any type of ranked genome-scale data. Dinu *et al.* [18] pointed out problems of GSEA and extended a widely used single gene analysis method (SAM [61]) to gene set analysis (SAM-GS). They employed the $L_2$ norm of the vector of the SAM statistics corresponding to the genes in the gene set as a statistic for further analysis.

Three types of null hypothesis are behind these different methods [23, 40, 60]. The first one assumes the same level of association of a gene set with the given phenotype as the complement of the gene set (Q1 or competitive). FATISCAN [1], CATMAP [10], and ERMINEJ [33] adopt a Q1 assumption. The second one assumes that there is no gene within a gene set associated with the phenotype (Q2 or self-contained). GLOBALTEST [24], SAFE [5], and SAM-GS [18] belong to the Q2 type. The third one is implemented by GSEA and states that none of the gene sets considered is associated with the phenotype (Q3). However, methods based on different null hypothesis may not lead to the same scientific conclusions. In a study of $p53$ wild-type versus mutant cancer cell lines [17], the self-contained methods identified many gene sets linked to $p53$ that are missed by competitive methods. Goeman and Bülmann [23] strongly opposed the testing of competitive null hypotheses with the use of gene sampling methods because of its unreasonable statistical independence assumption across genes. Based on a simulation study, Nam and Kim [40] recommended that these three type of methods should be used according to the goal of specific studies. If the purpose is to find gene sets relatively enriched with differentially expressed genes then a competitive method based on Q1 should be used. On the other hand, if the purpose is to find gene sets clearly separated between the two sample groups, then a self-contained method based on Q2 should be selected. In a general case, the Q3-based approach GSEA can avoid the clear drawbacks of the other methods and should be suggested.

Another debatable issue is about how to compute the $p$-value. There are two types of permutation test methods with respect to gene and sample randomization, for estimating the null hypothesis distribution. As pointed out by Goeman and Bühlman [23], gene randomization does not consider the correlation structures among functionally related genes and reverses the role of samples and genes in classical statistical tests. On the other hand, sample randomization requires a certain number of samples to make statistical inferences that are not applicable in many cases, such as time course data. Moreover, sample randomization-based methods often output too many gene sets as significant [60]. Therefore, Tian *et al.* [60] suggested that both

**Table 30.2 Database of functional groups of genes**

| Name | Gene Set Categories | URL |
|---|---|---|
| ASSESS | Cytogenetic, pathway, motif | http://people.genome.duke.edu/∼jhg9/assess/genesets.shtml |
| ErmineJ | GO | http://www.bioinformatics.ubc.ca/microannots/ |
| GAzer | GO, composite GO, InterPro, pathways, transcription factor binding site (TFBS) | http://integromics.kobic.re.kr/GAzer/document.jsp |
| GSA | Tissue, cellular processes, chromosome arms, cancer module, 5Mb chromosomal tiles, cytobands | http://www-stat.stanford.edu/∼tibs/GSA/ |
| MSigDb | Cytobands, curated pathways, computed | http://www.broad.mit.edu/gsea/msigdb/msigdb_index.html |
| PLAGE | Kyoto Encyclopedia of Genes and Genomes (KEGG) and BioCarta pathways | http://dulci.biostat.duke.edu/pathways/misc.html |

methods should be applied because the null hypotheses of them are different but complementary to each other.

The gene set database constructed by different methods are also important data sources for biological research (see Table 30.2). The known functional gene groups are collected based on diverse sources of biological knowledge such as GO, cytogenetic bands, pathways, cis-acting regulatory motifs, coregulated genes, and computational predicted gene modules. It should be noted that because the biological knowledge is incomplete, the gene set database contains limited functional gene sets that affects the performance of the gene set analysis methods.

### 30.2.2 Condition-Specific Pathway Inference

By integrating a microarray experiment dataset and prior established biologic knowledge, gene set analysis approaches can identify phenotype- or disease-related differentially expressed gene sets. These methods can reveal subtle but coordinated gene expression changes within a pathway. However, a major drawback is that they cannot discover new pathways correlated to phenotypes or diseases that have no records in pathway databases. To identify pathways' or modules' response to phenotypes, diseases, or changing conditions, many computational methods [9, 19, 26, 29, 39, 46, 47, 62, 71] have been developed by integrating protein–protein interaction, protein-DNA interaction, or metabolic networks with gene expression data (see Table 30.3). From a systematic perspective, genome-wide scanning functional pathways can overcome the limitation of local analysis and capture the cellular activities globally.

The principle of these methods is to model the phenotype related or condition-specific pathways or modules as connected subnetworks in the biomolecular interaction networks with significant expression changes between different conditions (*i.e.*, the active regions of the network). This type of methods generally include two steps

**Table 30.3 Methods of inferring condition-specific pathways**

| Ref. | Scoring Methods | Search Methods | Network Types |
|---|---|---|---|
| [29] | Gene based | Simulated annealing | Protein–protein interaction,<br>Protein–DNA interaction |
| [53] | Gene, interaction based | EM algorithm | Protein–protein interaction |
| [9] | Gene based | Greedy search | Gene Ontology relation,<br>Metabolic interaction |
| [56] | Gene based | Greedy search | Protein–protein interaction,<br>Transcriptional regulatory,<br>Literature mined |
| [35] | Interaction based | Enumerating and ranking | Protein–protein interaction |
| [12] | Interaction based | Hueristic algorithms | Protein–protein interaction,<br>Metabolic reactions,<br>Coexpression in regulons |
| [47] | Gene based | Heuristic algorithm | BioCarta pathways |
| [51] | Interaction based | Randomized algorithm | Protein–protein interaction |
| [26] | Interaction based | Simulated annealing | Protein–protein interaction |
| [39] | Gene based | Greedy search | Protein–protein interaction |
| [65] | Gene, interaction based | ICM algorithm | KEGG transcriptional pathways |
| [34] | Gene based | Simulated annealing | Protein–protein interaction |
| [13] | Gene based | Heuristic algorithm | Protein–protein interaction |
| [43] | Interaction based | Integer linear programming | Protein–protein interaction,<br>Protein–DNA interaction |
| [6] | Interaction based | Enumerating and ranking | Protein–protein interaction |
| [2] | Interaction based | Combine optimization | Protein–protein interaction |
| [62] | Gene based | Approximation algorithms | Protein–protein interaction |
| [64] | Gene, interaction based | Nonlinear programming | Protein–protein interaction |
| [19] | Gene based | Integer linear programming | Protein–protein interaction |
| [50] | Gene based | Heuristic algorithm | KEGG pathway |
| [71] | Interaction based | Integer linear programming | Protein–protein interaction |
| [46] | Gene based | Mixed integer programming | Protein–protein interaction |

(see Figure 30.1). In the first step, a scoring scheme to evaluate a module's active level is adopted based on each gene's or interaction's active level obtained from gene expression data. A different scoring function describes the different features of pathways. The gene-based scoring function emphasizes the expression changes of genes in a specific biological process, whereas the interaction-based methods focus on the specific cooperative patterns among proteins. Meanwhile, the group-based scoring function and the probabilistic model consider the active level both of genes and interactions to measure condition-specific pathways. In the second step, a search procedure is implemented to find the active pathways from a molecular interaction network such that the nodes in the active pathway are connected in the network. Unfortunately, this procedure is a nondeterministic polynomial (NP)-hard problem as was proved in [29]. Several fast heuristic methods are proposed to deal with this problem (*e.g.*, simulated annealing and greedy search), which often easily get stuck at a local optimal solution. Recently, several exact mathematic programming approaches were developed for finding condition-specific pathways. In terms of the scoring function, we can classify these methods into gene, interaction, group and probability-based methods, and according to the searching algorithms, they can divided into heuristic and mathematical programming-based methods. Here, we focus on the scoring function because different scoring functions define pathways with different characteristics; meanwhile, the heuristic search approaches are discussed. Particularly, the mathematical programming-based search methods are introduced separately because of its novelty and efficiency.

***30.2.2.1 Gene-Based Methods.*** The concept of active pathways is firstly introduced by Ideker *et al.* [29]. An active pathway means a connected region of a network that shows significant changes in expression over particular subsets of conditions. They designed a scoring scheme based on genes' activity to model the active subnetworks. Each gene's active level is measured by a $z$-score $z_i = \Phi^{-1}(1 - p_i)$, where $\Phi^{-1}$ is the inverse normal cumulated distribution function, and $p_i$ is the significant $p$-value from the single gene expression change analysis that represents the significant level of differential expression. For a subnetwork $A$ of $k$ genes, an aggregated $z$-score is defined as follows:

$$z_A = \frac{1}{\sqrt{k}} \sum_{i \in A} z_i$$

This score is independent to subnetwork size and standard normal distributed. By comparing it with the background distribution, the $z$-score is further adjusted as

$$s_A = \frac{z_A - \mu_k}{\sigma_k}$$

where the $\mu_k$ and $\sigma_k$ is the mean and standard deviation of randomly sampled gene sets of size $k$ by the Monte Carlo approach, respectively. The corrected score $s_A$ measures the level at which the score $z_A$ of a subnetwork is higher than the expected score corresponding to a random set of genes. Then the problem of identifying active subnetworks is to find the highest scoring subnetworks. The authors provided a proof that finding the maximal scoring connected subnetwork is an NP-hard problem. Thus, they designed a simulated annealing approach for this optimization problem. Each gene in the network is associated with an "active/inactive" state, and the active subnetwork is induced by the active nodes. An arbitrary temperature is set to a high degree initially and decreases gradually at each iteration. At each iteration, the configuration of the state is changed randomly, and the score of the highest scoring active component is calculated. The new configuration is retained if the change increases the score. Otherwise, it is accepted with a probability from high to low along with the temperature. The iteration terminated until the temperature decreased to zero and then output the active subnetworks. The application of this procedure (JACTIVEMODULES) on yeast data identified significant subnetworks with good correspondence to known regulatory mechanisms. This method has been implemented as a plug-in of the popular biomolecular network analysis software CYTOSCAPE [54]. Besides, Liu *et al.* [34] extended this method by applying Fisher's test to refine the results of the method to identify subnetworks related to type 2 diabetes mellitus. Other authors [56, 47, 3] implemented the similar scoring function but with heuristic search methods for the same goal.

In contrast to this continuous manner, some methods adopted a discrete scoring function to measure the activity of a subnetwork. Breitling *et al.* [9] proposed a method to evaluate a gene's active level based on the rank of genes computed according to their differential expression level. Then they searched active subnetworks by iteratively extending subnetworks, which are initially from the local minima nodes, by including the neighboring nodes with the next highest rank. The extension process is terminated when there is no outside neighboring node with a rank lower than the previously defined value. This heuristic method was applied to the classic yeast diauxic shift experiment data with GO and metabolic network information and identified several processes related to the yeast starvation response. In [62], focusing on the clinical case-control gene expression data, the authors introduced a bipartite graph including the case nodes representing case samples and the protein–protein interaction network in which genes are connected to the case nodes if they are expressed differentially in the case samples. The goal is to find a connected subnetwork with the smallest possibility in which a large proportion of genes are expressed differentially. This is a set cover problem in combinatorial optimization. Several approximation algorithms are implemented for getting the best subnetworks compared with randomly generated networks. On real disease data, the resulted dysregulated pathways are compact and enriched with hallmarks of the diseases. This method is robust to the outliers and has only two parameters, which both are with intuitive biological interpretations.

This kind of approach is computationally efficient because of its heuristic searching strategies and the additive scoring function. In practical implementation, it is suitable for when the active pathway has significant expression changes in each gene. However, heuristic approaches cannot guarantee identifying the global optimal scoring subnetworks. Some often output large high-scoring networks, which may be difficult to interpret. Additionally, the underlying assumption of the gene-based scoring function is that genes in an active subnetwork are independent, and the correlations among them are not considered in the scoring function.

***30.2.2.2 Interaction-Based Methods.*** Different from gene-based method, Guo *et al.* proposed an interaction-based method to extract active subnetworks responsive to conditions. The active level of a subnetwork is measured by the activity of interactions among proteins. The inactive interactions among the active proteins involved in the results of vertex-based methods are discarded. This method assigns a responsive score for each interactions in the network:

$$S(e_{12}) = \mathrm{Cov}(X_1, X_2) = \mathrm{Corr}(X_1, X_2)\mathrm{std}(X_1)\mathrm{std}(X_2)$$

where $e_{12}$ denotes the interaction between genes 1 and gene 2, $\mathrm{Corr}(X_1, X_2)$ is the Pearson correlation coefficient of the expression profiles of the two genes, $\mathrm{std}(X_1)$ and $\mathrm{std}(X_2)$ are the standard deviations of the expression profiles. The score for a connected subnetwork $A = (V, E)$ is defined as follows:

$$T(A) = \sum_{e \in E} S(e)$$

To eliminate the effect of $T$ by the number of the edges, the standardized score is defined as follow

$$S(A) = \frac{T(A) - \mathrm{avg}_k}{\mathrm{std}_k}$$

where $\mathrm{avg}_k$ and $\mathrm{std}_k$ are the mean and the standard deviation of randomly sampled subnetworks of $k$ edges, respectively. The numerical validation results show that the scores of subnetworks with different number of edges roughly follow the same distribution and are generally comparable. Similar to [29], the simulated annealing approach is used to find the maximal scoring subnetwork by iteratively altering the active state of each edge to the best configuration. The method was applied to human prostate cancer data and to yeast cell cycle data and efficiently can capture relevant protein interaction behaviors under the investigated conditions.

By integrating more known information, several interaction scoring methods [35, 2, 51, 6] have been developed for detecting signaling transduction networks between the known membrane receptor proteins and the transcription factor proteins, which can explain the molecular response mechanism of cells to the extracellular stimuli. Liu and Zhao [35] proposed a method for predicting the order of signaling pathway components via a score function that integrates protein–protein interaction data and gene expression data. All pathways whose molecular components are permuted are ranked according to the coexpression level or false negative rate of involved interactions. Only using the protein–protein interaction information, Scott *et al.* [51] applied an efficient method—color coding—for finding signaling paths and trees in a protein interaction network in which interactions are weighted by reliability based on experimental evidence. By randomly assigning colors to each protein, finding paths and trees starting from the receptor or a predefined protein set can be achieved by a dynamic programming with linear time complexity. To eliminate the bias resulted from false positives in the protein–protein interaction data, Bebek and Yang [6] combine the network with microarray expression, protein subcellular localization, and sequence information to weigh the interactions. They suggested a framework for identifying signalling pathways from biologically significant pathway segments.

The interaction-based methods model the active pathways as cascades of responsive interactions and can uncover the exact molecular interaction mechanism response to the phenotype or environmental conditions. They can be applied to temporal gene expression data to find specific biological process related subnetworks [26], whereas the node-based methods cannot. However, the protein–protein interaction and gene expression data obtained from a high-throughput experiment contain a lot of noise that may affect the results of these methods. In addition, they also suffer from the problems of local optimum caused by the inefficient searching methods, and the enumerative strategies are computationally intensive [51].

***30.2.2.3 Group-Based Methods.*** Following the ideal of Ideker *et al.* [29], Nacu *et al.* [39] proposed a new method that employs a different type of scoring function. The new scoring function is based on averaging expression profiles of all genes, not on the averaging score of each gene in a subnetwork. For a subnetwork including $k$ genes, its group expression is calculated as follows:

$$S_j = \sum_{i=1}^{k} X_{ij}$$

where $X_{ij}$ is the expression level for gene $i$ on array $j$. The activity measurement of the subnetwork is defined as the $t$-statistic:

$$f(S) = \frac{\mu_{i1} - \mu_{i0}}{\sqrt{\sigma_{i1}^2/n_1 + \sigma_{i0}^2/n_0}}$$

where the mean and standard deviation $\mu_{i1}$ and $\sigma_{i1}$ are for the set $\{S_j\}$, where $j$ is a case, and $\mu_{i0}$ and $\sigma_{i0}$ are for $\{S_j\}$, where $j$ is a control. This scoring function takes into account the correlation structure among genes that are ignored by the methods based on averaging testing statistics of individual genes. The high-scoring subnetwork are extracted from the whole network by a greedy search procedure that starts with a seed node and picks a node such that the new subnetwork has a maximal score in each step until the subnetwork size is to a predefined number or until other stopping criterions are met. This method was applied to human cancer and immune system data and retrieved several known and potential pathways.

In [13], Chuang *et al.* proposed a method for finding breast cancer metastasis-related subnetwork biomarkers. For a given subnetwork, its active score vector $a'$ is derived by a $z$-transform from the normalized gene expression values over samples. The discriminative or active score is defined as the mutual information between $a'$ and the sample label vector $c$,

$$S = \sum_{x_1 \in a'} \sum_{x_2 \in c} p(x_1, x_2) \log \frac{p(x_1, x_2)}{p(x_1)p(x_2)}$$

A greedy search is performed to identify candidate subnetworks with local maximal scores. Using the subnetwork as features, a logistic regression model was trained for classifying the metastasis and nonmetastasis tumors. In the two case studies on breast cancer data, the subnetwork markers are reproducible and achieve high classification accuracy.

The group-based scoring function considered the correlation among all genes in the subnetworks. It may be more precise to consider the correlation according to the topological structure of a network. The concept of a network biomarker provides many insights into the biological mechanisms by mining information from biomolecular interaction networks.

***30.2.2.4 Probabilistic Models.*** In [53], Segal *et al.* proposed an unified probabilistic model to identify pathways with two properties. One is that genes in a pathway exhibit a similar gene expression profile, and the other is that the protein products of the genes often interact with each other. The combined model defines a joint distribution over the hidden variables $C = \{c_1, ..., c_n\}$ as follows:

$$P(C, X|E) = \frac{1}{Z}\left(\prod_{i=1}^{n} P(c_i)\prod_{j=1}^{m} P(X_{ij}|c_i)\right)\left(\prod_{e_{ij}\in E} \Phi_2(c_i, c_j)\right)$$

where $X$ represents the gene expression profiles, $E$ represents the protein–protein interactions, $Z$ is a normalizing constant that ensures that $P$ adds up to 1. The hidden variable $c_i \in C$ represents the cluster to which gene $i$ belongs to. Given the number of cluster $k$, the variable $c_i$ is associated with a multinormial distribution with parameters $\Theta = \{\theta_1, \ldots, \theta_k\}$. $P(X_{ij}|c_i)$ is the conditional probability distribution that represents the probability of observing an expression value $X_{ij}$ at the $j$th condition of the expression profile of gene $i$, assuming gene $i$ belongs to cluster $c_i$. The model assumes that $P(X_{ij}|c_i)$ follows a Gaussian distribution $N(\mu_{c_i j}, \sigma^2_{c_i j})$. Each pair of genes $i$ and $j$ that interact in $E$ are associated with a compatibility potential $\Phi_2(c_i, c_j)$ in which $\Phi_2(c_i, c_j) = \alpha$ if $c_i = c_j$ and $\Phi_2(c_i, c_j) = 1$ otherwise, where $\alpha$ is a parameter and required to be greater than or equal to 1. Finally, this probabilistic model is learned by an EM algorithm [16]. The results on yeast data show that this method can uncover coherent functional groups and biologically significant protein complexes.

Other probabilistic models, such as the Markov random field model [65] and the mixture model [50], also are employed for identifying disease-related genes and subnetworks. All former probabilistic methods use the structural information of the interaction networks to capture the dependency of differential expression patterns for genes in the network. They are both based on the theoretical rigorous probabilistic model that has good theoretical properties. However, there are still some problems for practical purpose. For example, real biological data may not follow the predefined probability distributions used in these models, the parameters are hard to estimate, and the computational cost is high for large-scale networks.

***30.2.2.5 Mathematical Programming Methods.*** The powerful mathematical programming methods have been well developed in the operational research field in the past 60 years. As a typical optimization problem for extracting active pathways, mathematical programming can be a promising tool to solve it. Qiu *et al.* [46] proposed a new computational method to detect condition-specific pathways or to identify differentially expressed pathways via the integration of gene expression and interactomic data in a sophisticated and efficient manner. The signal-to-noise ratio measuring a differentially expressed level is implemented as the active score for each gene. The active level of a subnetwork is defined as the mean of the active scores of the involved genes. Then, given a predefined root gene, the active subnetwork connecting the root gene is extracted from the molecular interaction network by the following mixed integer linear programming model:

$$\begin{aligned}
\max \quad & \frac{1}{R}\sum_{i=1}^{n} t_i x_i \\
\text{s.t.} \quad & \sum_j y_{1j} = R - 1 \\
& \sum_j y_{ji} - \sum_{j \neq 1} y_{ij} = x_i, \quad i = 2, \ldots, n \\
& y_{ij} \leq (R-1)x_i, \quad i, j = 1, \ldots, n \\
& x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n
\end{aligned}$$

where $t_i$ represents the active score, $x_i$ represents whether gene $i$ is selected ($x_i = 1$) or not ($x_i = 0$) into the subnetwork; the root gene is labeled as 1, $y_{ij}$ are dummy variables representing the flow between selected nodes, $R$ is a predefined constant, which is the size of the active subnetwork, and $n$ is the total number of proteins in the network. The constraints ensure the connectivity of the selected nodes, which is a major advantage of the proposed method. Consider each gene in the network as the root genes; the subnetworks with a maximal active score can be identified. To select the significant ones among them, the density distributions of the scores of the identified subnetworks are estimated using a nonparameter kernel density estimation method, and a percentile is calculated to distinguish the significant subnetworks. Finally, all significant subnetworks are mapped to the original network to generate an integrated differentially expressed pathway. The results on yeast and human data demonstrate that this method is more accurate and robust than the simulated annealing based approach.

The same as the previous approach emphasizing the connectivity of members in the condition-specific pathway, Dittrich *et al.* [19] proposed an exact approach to identify functional modules in protein–protein interaction networks. By considering the distribution of the experimentally derived $p$-values corresponding to an expression change of genes as a mixture of a noise and a signal component [45], each gene is scored by an additive likelihood ratio of signal to noise. The active score of a subnetwork is given by the sum of scores of all involved genes. Then, a combinatorial optimization method [36] for the prize-collecting Steiner tree problem was used to find the maximal scoring subnetworks. This approach can obtain exact solutions even when the network includes 2034 nodes. Applied to the lymphoma microarray data, this method found several modules, and one of them is associated biologically with proliferation overexpressed in the aggressive activated B-cell (ABC) subtype.

Considering the interactions in the pathways, Wang and Xia [64] suggested an optimization framework to identify condition-specific subnetworks, which are enriched with condition-specific genes and interactions between them. A continuous optimization model is designed as follows:

$$\begin{aligned} \max \quad & \sum_i \sum_j w_{ij} x_i x_j + \lambda \sum_i f_i x_i \\ \text{s.t.} \quad & x_1^\beta + x_2^\beta + \cdots + x_n^\beta = 1 \\ & x_i \geq 0, \qquad i = 1, 2, \cdots, n \end{aligned}$$

where $w_{ij}$ represents the weight of interactions between genes, $x_i$ can be interpreted as whether the $i$th node is included in the condition specific subnetwork, and $f_i$ represents the association level between the gene and specific conditions. The first term in the objective function measures the interconnectivity within the subnetwork, whereas the second term measures the degree of association between the subnetwork nodes and the specific condition. The constraints regulate the number of nodes in the final subnetworks by parameter $\beta$. Because the continuous optimization problem is easier to solve, this approach is fast and has been applied to identify type 2 diabetes-related subnetworks.

Recently, mathematical programming methods were applied to identify signaling transduction networks [71, 2, 43]. Zhao *et al.* [71] proposed a integer linear programming model by integrating protein–protein interaction and gene expression data as follows:

$$\begin{aligned} \min \quad & -\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} y_{ij} + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} y_{ij} \\ \text{s.t.} \quad & y_{ij} \leq x_i \\ & y_{ij} \leq x_j \\ & \sum_{j=1}^{n} y_{ij} \geq 1, \text{ if } i \text{ is either a starting or ending protein} \\ & \sum_{j=1}^{n} y_{ij} \geq 2x_i, \text{ if } i \text{ is not a starting or ending protein} \\ & x_i = 1, \text{ if } i \text{ is a protein known in the signaling pathway} \\ & x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n \\ & y_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \ldots, n \end{aligned}$$

where $w_{ij}$ is the weight of interaction, which is the confidence score of the interaction or the expression correlation coefficient based on gene expression data, and $x_i$ and $y_{ij}$ are binary variables to denote whether protein $i$ and the interaction are selected as a component of the signaling pathway, respectively. The first term of the objective function aims to find a maximal weighted pathway, whereas the second term is used to control the size of the pathway. A tradeoff exists between the two terms, which can be balanced by the parameter $\lambda$. The starting proteins and ending proteins are receptors and transcription factors, respectively. The solution is obtained by a relaxed linear programming algorithm. The numerical results on yeast mitogen-activated protein kinase (MAPK) signaling pathways demonstrate that the proposed model can efficiently uncover pathways, and the prediction results were in high agreement with the current biological knowledge.

The mathematical programming methods significantly enhance the searching procedure by guaranteeing the optimality of solutions in a general case without significantly increasing the computational complexity. The solution space of the mathematical programming model can be interpreted as polyhedra in a high-dimensional space. Theoretical analysis of these results often leads to new insights into understanding the original problems [19]. As a typical optimization problem, some well-studied models and algorithms can also be applied to identify active pathways from high-throughput data.

Considering phenotype association in the sequence level, many methods modeling genetic pathways are proposed. In [68], by integrating genetic and gene expression data into the protein–protein network, a flow-based algorithm is proposed to reveal cellular pathways responding to alpha-synuclein toxicity. By examining more than 150 distinct stimuli, the author found that differentially expressed genes and genetic hits, genes whose individual manipulation alters the phenotype of stimulated cells, are consistently disparate. The identified sparse subnetworks connecting genetic hits to the differentially expressed genes can explain the relationship between alpha-synuclein toxicity and basic cellular pathways. Using a similar network flow model, Suthram *et al.* [59] presented an approach called eQED to identify the causal factors responsible for the observed expression changes. The candidate casual genes first are obtained from the analysis of the expression quantitative trait loci (eQTLs). The eQED use the protein–protein interaction network to provide regulatory pathways from the casual gene to the differentially expressed genes. In another research work [4], single nucleotide polymorphisms (SNPs) association scores are mapped onto the proteins of human protein interaction networks. Subnetworks containing a higher proportion of genes associated with the disease than expected by chance are extracted. The implementation of this approach on multiple sclerosis data first identified neural pathways related to multiple sclerosis. In summary, the genetic pathways can uncover the relationship of genetic interaction by information flow between proteins that cannot be detected in the transcription level. It is easy to see that the network flow model is used widely for pathway modeling because it is an efficient mode to guarantee the connectivity and is easy to solve.

This section surveys the present methods for inferring condition-specific pathways. Many of these methods are implemented and can be downloaded as software (Table 30.4). All these methods are not restricted on the type of molecular interaction networks. Thereby, they also can be applied to other types of networks such as metabolic, transcription regulation networks to extract high related regions

**Table 30.4 Softwares for inferring condition-specific pathways**

| Software | Year | Ref. | URL |
|---|---|---|---|
| JACTIVEMODULES | 2002 | [29] | http://chianti.ucsd.edu/cyto_web/plugins |
| GIGA | 2004 | [9] | http://www.biomedcentral.com/content/ supplementary/1471-2105-5-100-S1.exe |
| TOPNET | 2004 | [56] | http://www.biosolveit.de/topnet/ |
| GXNA | 2007 | [39] | http://stat.stanford.edu/ serban/gxna |
| GNEA | 2007 | [34] | from the author upon request |
| SPINE | 2007 | [43] | http://cs.tau.ac.il/ roded/SPINE.html |
| NET-SYNTHESIS | 2007 | [2] | http://www.cs.uic.edu/ dasgupta/network-synthesis/ |
| HEINZ | 2008 | [19] | http://www.planet-lisa.net |
| MMG | 2008 | [50] | http://www.dcs.shef.ac.uk/ guido/software.html |
| MILPS | 2009 | [46] | http://zhangroup.aporc.org/YuqingQiu |

corresponding to the conditions or phenotypes. Meanwhile, the new identified condition-specific pathways can be included in the gene set database to alleviate the limitations of gene set analysis methods caused by the incompleteness of known functional gene sets. Obviously, the well-established statistics adopted in single gene and gene set analysis methods can be considered to be incorporated into the mathematical programming method. However, scoring functions may become more complicated, and thereby more difficult to solve because of the NP-hard nature of this problem even for simple scoring functions. Hence, designing an accurate and easy methodology for solving this problem is still an important direction.

## 30.3 DISEASE GENE PRIORITIZATION AND GENETIC PATHWAY DETECTION

The condition-specific pathway identification methods can uncover the gene's function and relationship between them. The ultimate goal is to find the most likely genes that could be verified by experiment, especially for disease. Deciphering genes associated with specific diseases is an important task to discover the genetic mechanism in the molecular level and has a potential role in biomedical research.

Traditional DNA sequence-based gene-disease association analysis approaches, such as linkage analysis or association studies [8], successfully have identified disease associated chromosomal regions, each of which contains hundreds of genes. Experimental examination of causative mutations for all these genes is time consuming and expensive. It is often impossible to identify the correct disease gene by inspecting of the list of genes within the region. On the other hand, various "omics" data and other data sources, such as protein–protein interaction, protein structure, GO functional data, gene expression data, as well as published literatures, are generated and cumulated in the postgenomic era. Computational methods to mine these data for disease pathways identification and candidate disease prioritization become important topics. Hence, we extend the condition-specific pathway identification to the latest development of disease gene prioritization and other highly related genetic pathway detection issues in this section.

Besides a small part of genetic diseases that are contributed by single genes, most common genetic disorders have a complex inheritance and may result from variants in many genes, each contributing only weak effects to the disease. By various biological experiments, many disease genes have been validated and are recorded in the Online Mendelian Inheritance in Man (OMIM) database [27]. The studies [25, 41] on the properties of known disease genes pointed out the modularity nature of the disease genes that are phenotypically similar diseases and often are caused by functionally related genes (as shown in Figure 30.2). A biological module of disease genes responsible for a specific disease could be a protein complex, a biological pathway, or a subnetwork of biomolecular interaction networks. With this understanding, a vast amount of disease gene prediction methods are proposed by integrating various information, such as biomolecular interaction networks, phenotype similarity networks, and so on. For a particular disease, candidate genes are ranked based on their similarity to the known genes related to the disease for further experimental examination.

Using the protein–protein interaction network or gene functional linkage network, candidate gene prioritization simply can be done by searching for direct neighbors of known disease genes [42] or by calculating the shortest path between candidates and know disease proteins [22]. In contrast to these local similarity measures, global

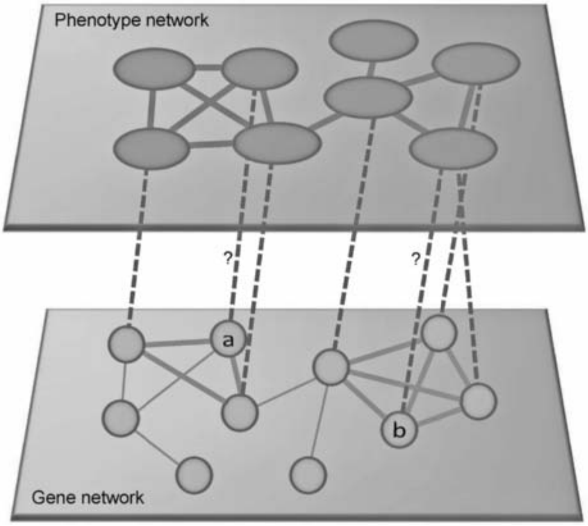


**Figure 30.2** A gene-phenotype network. In this hypothetical example, some diseases in the phenotype network have known causative genes, and some genes are related phenotypically to one or more disease. But some diseases lack identified causative gene(s). If the known causative genes functionally are closely related, then candidate genes (gene a and b) can be hypothesized to be corresponding to disease because of their close functional relationships to the known genes of the phenotypically related diseases.

network similarity measures—random walk and diffusion kernel—are proposed [31]. The random walk on graphs is defined as an iterative walker's transition from its current node to a randomly selected neighbor starting at a given source node. A variant of random walk that is used for disease gene ranking is defined in [31] as follows:

$$p^{t+1} = (1 - r)Wp^t + rp^0$$

where $W$ is the column-normalized adjacency matrix of the graph and $p^t$ is a vector in which the $i$th element holds the probability of being at node $i$ at time step $t$. The initial probability vector $p^0$ is constructed such that equal probabilities are assigned to the known disease protein, with the sum probabilities equal to 1. Candidate genes are ranked according to the values in the steady-state probability vector $p^{\infty}$. Another similarity measurement—diffusion kernel $K$—of a network is defined as

$$K = e^{-\beta L}$$

where $\beta$ controls the magnitude of the diffusion. The matrix $L$ is the Laplacian of the graph, defined as $D - A$, where $A$ is the adjacency matrix of the interaction network and $D$ is a diagonal matrix containing the nodes' degrees. The rank for each gene $j$ is assigned in accordance with its score defined as follows:

$$\text{score}(j) = \sum_{i \in \text{disease genes}} K_{ij}$$

In the simulated test examples [31], candidate genes are selected from the simulated linkage intervals of 100 genes surrounding the disease gene. The global measures achieved an area under the receiver operating characteristic (ROC) curve of up to 98% and significantly outperform local distance measures methods.

By integrating the phenotype similarity network, phenotype-gene association data, and protein–protein interaction networks, computational models [32, 66] have been developed for interpreting the relationship between disease gene modules and phenotypes and, furthermore, for predicting unknown disease genes. In [32], Lage *et al.* proposed a Bayesian model for prioritizing protein complexes that are predicted from the protein–protein interaction network. Each candidate gene is ranked by the phenotype similarity score of the complex containing it. The assumption underlying this model is that mutations in different members of a protein complex lead to similar (or identical) disorders. Thus, a protein likely is to be involved in the molecular pathology of a disorder if it is in a high-confidence complex in which some proteins are known to be associated with similar disorders. In another work, Wu *et al.* [66] suggested that the modularity of disease genes indicates a positive correlation between gene–gene relationship and phenotype–phenotype similarity. They proposed a regression model that explains the phenotype similarity by gene closeness in the molecular interaction network and then uses the correlation between phenotype similarities and gene closeness to predict disease genes. Because there is still no causative gene identified for some diseases, integration of phenotype data can provide novel genes implicated in the diseases.

The modular nature of disease genes as well as the idea that directly or indirectly interacting proteins cause similar disorders are important in elucidating the molecular mechanism of human genetical disorders and uncovering related genes. Although integration of heterogeneous data can greatly improve the accuracy of candidate gene prioritization and pathway identification, the available biological data, such as high-throughput experimental data, are incomplete and noisy. Additionally, the measure of biological information such as the similarity of phenotypes do not have a standard to evaluate quantitatively. The integration framework can alleviate the noise and make more robust and reliable prediction and identification.

## 30.4 MODULE NETWORKS

Modules are the building blocks of biological molecular interaction networks in a high level. The activity of a cell is organized as a network of interacting modules, such as condition-specific pathways, groups of genes of the same cellular functions, and sets of genes coregulated to respond to different conditions. Experimental evidence [67] has showed that modules are assembled to enable coordinated and complex functions. The identification of modules and the intermodule relationships enables the understanding of cellular behavior.

The first stage of the analysis of a module network is to identify modules and then analyze the relationship between them. Many computational methods have been proposed based on the protein–protein interaction networks or gene expression data. The biomolecular network based approaches [55, 70, 69] define modules as subnetworks that are connected densely within themselves but are connected sparsely with the rest of the network. The identified modules correlate well with protein complexes identified experimentally and tend to contain proteins with similar functions. Another kind of approach uses the gene expression data [37] for extracting modules that are groups of genes with similar expression profiles. The motivation behind this strategy is the assumption that coexpressed genes are coordinately regulated. However, a functionally unrelated gene also could have a similar expression behavior. Much prior knowledge, such as regulator–target gene interaction and known pathways, can be integrated with expression data to reveal the module networks. For instance, Segal *et al.* [52] present a probabilistic method for identifying regulatory modules and the regulation programs among them from gene expression data coupled with regulator data. They implemented their method on yeast expression data and obtained functionally coherent modules and their correct regulators. As mentioned in the previous sections, single source data have limitations because they contain many false positives and are incomplete. Integrating protein–protein interaction data with gene expression data or another kind of data can identify more reliable modules.

Another possible way is to employ the condition-specific pathway identification methods for this problem and then construct more reliable condition-specific module networks for special conditions. The analysis of the condition-specific module network can shed light on the cooperation of modules related to specific condition and uncover specific cooperative mechanisms. For example, the yeast MAPK and high osmolarity glycerol (HOG) pathways were connected by the Ste11 gene. The Ste11 crosstalks between these two pathways were proved by literatures. More quantitatively, the dynamic analysis of module networks can elucidate the molecular operations within the modules and the cooperations between the modules along with time or in different conditions, which will provide insights into the prediction and manipulation and eventually into the understanding of the cell activities *in vivo*.

## 30.5 SUMMARY

In this chapter, we surveyed the topics highly related with condition-specific pathway identification. The key methods include gene set analysis methods that integrate gene expression data and known functional gene set and condition-specific pathway inference methods by combining gene expression data with molecular interaction data. The gene set analysis methods are useful for high lighting known pathways with specific conditions but are constrained by the gene set database and cannot identify new pathways. The second kind of method can uncover condition-specific pathways from the biomolecular interaction networks and have promising applications in uncovering unknown interacting mechanisms. The extending survey on genetic pathway identification and disease gene prioritization studies enlarge the applications surveyed in the chapter, and the module network analysis can advance the understanding of biological systems in a global view.

## ACKNOWLEDGEMENTS

This work is supported partially by the National Natural Science Foundation of China under Grant No. 60873205, No. 10801131, by the Beijing Natural Science Foundation under Grant No. 1092011, by the Innovation Project of Chinese Academy of Sciences, kjcs-yw-s7 and by the Ministry of Science and Technology, China under Grant No. 2006CB503905.

## REFERENCES

1. F. Al-Shahrour, L. Arbiza, H. Dopazo, J. Huerta-Cepas, P. Minguez, D. Montaner, and J. Dopazo. From genes to functional classes in the study of biological systems. *BMC Bioinformatics*, 8(1):114, 2007.
2. R. Albert, B. DasGupta, R. Dondi, S. Kachalo, E. Sontag, A. Zelikovsky, and K. Westbrooks. A novel method for signal transduction network inference from indirect experimental evidence. *J Computat Biol*, 14(7):927–949, 2007.
3. S. Bandyopadhyay, R. Kelley, and T. Ideker. Discovering regulated networks during HIV-1 latency and reactivation. *Pacific Symposium on Biocomputing*, Volume 11, 2006, pp. 354–366.

4. S.E. Baranzini, N.W. Galwey, J. Wang, P. Khankhanian, R. Lindberg, D. Pelletier, W. Wu, B.M.J. Uitdehaag, L. Kappos, Gene MSA Consortium. Pathway and network-based analysis of genome-wide association studies in multiple sclerosis. *Hum Mol Genet*, 18(11):2078–2090, 2009.
5. T.W. Barry, B.A. Nobel, and A.F. Wright. Significance analysis of functional categories in gene expression studies: A structured permutation approach. *Bioinformatics*, 21(9):1943–1949, 2005.
6. G. Bebek and J. Yang. Pathfinder: Mining signal transduction pathway segments from protein-protein interaction networks. *BMC Bioinformatics*, 8(1):335, 2007.
7. Y. Ben-Shaul, H. Bergman, and H. Soreq. Identifying subtle interrelated changes in functional gene categories using continuous measures of gene expression. *Bioinformatics*, 21(7):1129–1137, 2005.
8. D. Botstein and N. Risch. Discovering genotypes underlying human phenotypes: Past successes for mendelian disease, future approaches for complex disease. *Nat Genet*, 33(3s):228–237, 2003.
9. R. Breitling, A. Amtmann, and P. Herzyk. Graph-based iterative group analysis enhances microarray interpretation. *BMC Bioinformatics*, 5(1):100, 2004.
10. T. Breslin, P. Edén, and M. Krogh. Comparing functional annotation analyses with Catmap. *BMC Bioinformatics*, 5:193, 2004.
11. P.O. Brown and D. Botstein. Exploring the new world of the genome with dna microarrays. *Nat Genet*, 21(1 Suppl):33–7, 1999.
12. L. Cabusora, E. Sutton, A. Fulmer, and V.C. Forst. Differential network expression during drug and stress response. *Bioinformatics*, 21(12):2898–2905, 2005.
13. H.Y. Chuang, E. Lee, Y.T. Liu, D. Lee, and T. Ideker. Network-based classification of breast cancer metastasis. *Mol Syst Biol*, 3(140), 2007.
14. The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nat Genet*, 25:25–29, 2000.
15. K.D. Dahlquist, N. Salomonis, K. Vranizan, S.C. Lawlor, and B.R. Conklin. GenMAPP, a new tool for viewing and analyzing microarray data on biological pathways. *Nat Genet*, 31(1):19–20, 2002.
16. A.P. Dempster, N.M. Laird, D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J Roy Stat Soc*, 39(1):1–38, 1977.
17. I. Dinu, D.J. Potter, T. Mueller, Q. Liu, J.A. Adewale, S.G. Jhangri, G. Einecke, S.K. Famulski, P. Halloran, and Y. Yasui. Gene-set analysis and reduction. *Brief Bioinform*, 10(1):24–34, 2009.
18. I. Dinu, J. Potter, T. Mueller, Q. Liu, A. Adewale, G. Jhangri, G. Einecke, K. Famulski, P. Halloran, and Y. Yasui. Improving gene set analysis of microarray data by SAM-GS. *BMC Bioinformatics*, 8(1):242, 2007.
19. T.M. Dittrich, W.G. Klau, A. Rosenwald, T. Dandekar, and T. Muller. Identifying functional modules in protein-protein interaction networks: an integrated exact approach. *Bioinformatics*, 24(13):i223–231, 2008.
20. S.W. Doniger, N. Salomonis, K.D. Dahlquist, K. Vranizan, S.C. Lawlor, B.R. Conklin. MAPPFinder: Using Gene Ontology and GenMAPP to create a global gene-expression profile from microarray data. *Genome Biol*, 4(1):R7, 2003.
21. B. Efron and R. Tibshirani. On testing the significance of sets of genes. *Ann Appl Stat*, 1(1):107–129, 2007.

22. L. Franke, H. Bakel, L. Fokkens, E.D. de Jong, M. Egmont-Petersen, and C. Wijmenga. Reconstruction of a functional human gene network, with an application for prioritizing positional candidate genes. *Am J Hum Genet*, 78(6):1011–1025, 2006.

23. J.J. Goeman and P. Bühlmann. Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics*, 23(8):980–987, 2007.

24. J.J. Goeman, A.S. van de Geer, F. de Kort, and C.H. van Houwelingen. A global test for groups of genes: testing association with a clinical outcome. *Bioinformatics*, 20(1):93–99, 2004.

25. K.I. Goh, M.E. Cusick, D. Valle, B. Childs, M. Vidal, and A.L. Barabasi. The human disease network. *Proc Natl Acad Sci U S A*, 104(21):8685, 2007.

26. Z. Guo, Y. Li, X. Gong, C. Yao, W. Ma, D. Wang, Y. Li, J. Zhu, M. Zhang, D. Yang, et al. Edge-based scoring and searching method for identifying condition-responsive protein protein interaction sub-network. *Bioinformatics*, 23(16):2121, 2007.

27. A. Hamosh, A.F. Scott, J.S. Amberger, C.A. Bocchini, and V.A. McKusick. Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders. *Nucleic Acids Res*, 33(Suppl 1):D514–517, 2005.

28. L.H. Hartwell, J.J. Hopfield, S. Leibler, and A.W. Murray. From molecular to modular cell biology. *Nature*, 402(Supp 6761):C47, 1999.

29. T. Ideker, O. Ozier, B. Schwikowski, and A.F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(Suppl 1):233–240, 2002.

30. M. Kanehisa, S. Goto, M. Hattori, K.F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. Araki, and M. Hirakawa. From genomics to chemical genomics: new developments in KEGG. *Nucleic Acids Res*, 34(Database Issue):D354, 2006.

31. S. Köhler, S. Bauer, D. Horn, and P.N. Robinson. Walking the interactome for prioritization of candidate disease genes. *Am J Hum Genet*, 82(4):949–958, 2008.

32. K. Lage, E.O. Karlberg, Z.M. Størling, P.Í. Ólason, A.G. Pedersen, O. Rigina, A.M. Hinsby, Z. Tümer, F. Pociot, N. Tommerup, et al. A human phenome-interactome network of protein complexes implicated in genetic disorders. *Nat Biotechnol*, 25(3):309–316, 2007.

33. H.K. Lee, W. Braynen, K. Keshav, and P. Pavlidis. ErmineJ: Tool for functional analysis of gene expression data sets. *BMC Bioinformatics*, 6(1):269, 2005.

34. M. Liu, A. Liberzon, S.W. Kong, W.R. Lai, P.J. Park, and K. Kerr. Network-based analysis of affected biological processes in type 2 diabetes models. *PLoS Genet*, 3(6):e96, 2007.

35. Y. Liu and H. Zhao. A computational approach for ordering signal transduction pathway components from genomics and proteomics data. *BMC Bioinformatics*, 5(1):158, 2004.

36. I. Ljubić, R. Weiskircher, U. Pferschy, G.W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Math Program*, 105(2):427–449, 2006.

37. D.J. Miller, Y. Wang, and G. Kesidis. Emergent unsupervised clustering paradigms with potential application to bioinformatics. *Front Biosci*, 13:677, 2008.

38. V.K. Mootha, C.M. Lindgren, K.F. Eriksson, A. Subramanian, S. Sihag, J. Lehar, P. Puigserver, E. Carlsson, M. Ridderstrale, E. Laurila, *et al.* PGC-1alpha-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nat Genet*, 34(3):267–73, 2003.

39. S. Nacu, R. Critchley-Thorne, P. Lee, and S. Holmes. Gene expression network analysis and applications to immunology. *Bioinformatics*, 23(7):850, 2007.

40. D. Nam and S.Y. Kim. Gene-set approach for expression pattern analysis. *Brief Bioinform*, 9(3):189–197, 2008.

41. M. Oti and H.G. Brunner. The modular nature of genetic diseases. *Clinl Genet*, 71(1):1–11, 2007.

42. M. Oti, B. Snel, M.A. Huynen, and H.G. Brunner. Predicting disease genes using protein–protein interactions. *J Med Genet*, 43(8):691–698, 2006.

43. O. Ourfali, T. Shlomi, T. Ideker, E. Ruppin, and R. Sharan. SPINE: A framework for signaling-regulatory pathway inference from cause-effect experiments. *Bioinformatics*, 23(13):i359–366, 2007.

44. K.H. Pan, C.J. Lih, and N.S. Cohen. Effects of threshold choice on biological conclusions reached during analysis of gene expression by DNA microarrays. *Proc Natl Acad Sci U S A*, 102(25):8961–8965, 2005.

45. S. Pounds and W.S. Morris. Estimating the occurrence of false positives and false negatives in microarray studies by approximating and partitioning the empirical distribution of p-values. *Bioinformatics*, 19(10):1236–1242, 2003.

46. Y.Q. Qiu, S. Zhang, X.S. Zhang, and L. Chen. Identifying differentially expressed pathways via a mixed integer linear programming model. *IET Syst Biol*, 3(6):475-486, 2009.

47. D. Rajagopalan and P. Agarwal. Inferring pathways from gene lists using a literature-derived network of biological relationships. *Bioinformatics*, 21(6):788–793, 2005.

48. B. Ren, F. Robert, J.J. Wyrick, O. Aparicio, E.G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Hannett, E. Kanin, *et al.* Genome-wide location and function of DNA binding proteins. *Science*, 290(5500):2306–2309, 2000.

49. J.F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G.F. Berriz, F.D. Gibbons, M. Dreze, N. Ayivi-Guedehoussou, et al. Towards a proteome-scale map of the human protein-protein interaction network. *Nature*, 437(7062):1173–1178, 2005.

50. G. Sanguinetti, J. Noirel, and C.P. Wright. MMG: A probabilistic tool to identify submodules of metabolic pathways. *Bioinformatics*, 24(8):1078–1084, 2008.

51. J. Scott, T. Ideker, M.R. Karp, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J Comput Biol*, 13(2):133–144, 2006.

52. E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34(2):166–176, 2003.

53. E. Segal, H. Wang, and D. Koller. Discovering molecular pathways from protein interaction and gene expression data. *Bioinformatics*, 19(Suppl 1):264–272, 2003.

54. P. Shannon, A. Markiel, O. Ozier, N.S. Baliga, J.T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, 13(11):2498, 2003.

55. R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Mol Syst Biol*, 3(1):88, 2007.

56. F. Sohler, D. Hanisch, and R. Zimmer. New methods for joint analysis of biological networks and expression data. *Bioinformatics*, 20(10):1517–1521, 2004.

57. U. Stelzl, U. Worm, M. Lalowski, C. Haenig, F.H. Brembeck, H. Goehler, M. Stroedicke, M. Zenkner, A. Schoenherr, S. Koeppen, et al. A human protein-protein interaction network: a resource for annotating the proteome. *Cell*, 122(6):957–968, 2005.

58. A. Subramanian, P. Tamayo, V.K. Mootha, S. Mukherjee, B.L. Ebert, M.A. Gillette, A. Paulovich, S.L. Pomeroy, T.R. Golub, E.S. Lander, *et al*. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci U S A*, 102(43):15545–15550, 2005.
59. S. Suthram, A. Beyer, R.M. Karp, Y. Eldar, and T. Ideker. eQED: an efficient method for interpreting eQTL associations using protein networks. *Mol Syst Biol*, 4(1):162, 2008.
60. L. Tian, A.S. Greenberg, W.S. Kong, J. Altschuler, S.I. Kohane, and J.P. Park. Discovering statistically significant pathways in expression profiling studies. *Proc Natl Acad Sci U S A*, 102(38):13544–13549, 2005.
61. G. V. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proc Natl Acad Sci U S A*, 98(9):5116–5121, 2001.
62. I. Ulitsky, R.M. Karp, and R. Shamir. Detecting disease-specific dysregulated pathways via analysis of clinical expression profiles. *Lect Notes Comput Sci*, 4955:347, 2008.
63. X. Wang, E. Dalkic, M. Wu, and C. Chan. Gene module level analysis: identification to networks and dynamics. *Curr Opin Biotech*, 19(5):482–491, 2008.
64. Y. Wang and Y. Xia. Condition specific subnetwork identification using an optimization model. *Proceedings of 2nd International Symposium on Optimization and Systems Biology, Lecture Notes in Operations Research*, volume 9, 2008, pp. 333–340.
65. Z. Wei and H. Li. A Markov random field model for network-based analysis of genomic data. *Bioinformatics*, 23(12):1537–1544, 2007.
66. X. Wu, R. Jiang, M.Q. Zhang, and S. Li. Network-based global inference of human disease genes. *Mol Syst Biol*, 4(1):184, 2008.
67. I. Yanai, L.R. Baugh, J.J. Smith, C. Roehrig, S.S. Shen-Orr, J.M. Claggett, A.A. Hill, D.K. Slonim, and C.P. Hunter. Pairing of competitive and topologically distinct regulatory modules enhances patterned gene expression. *Mol Syst Biol*, 4(1):163, 2008.
68. E. Yeger-Lotem, L. Riva, L.J. Su, A.D. Gitler, A.G. Cashikar, O.D. King, P.K. Auluck, M.L. Geddie, J.S. Valastyan, D.R. Karger, et al. Bridging high-throughput genetic and transcriptional data reveals cellular responses to alpha-synuclein toxicity. *Nat Genet*, 41(3):316–323, 2009.
69. S. Zhang, G. Jin, X.-S. Zhang, and L Chen. Discovering functions and revealing mechanisms at molecular level from biological networks. *Proteomics*, 7(16):2856–2869, 2007.
70. S. Zhang, X. Ning, and X.S. Zhang. Identification of functional modules in a PPI network by clique percolation clustering. *Comput Biol Chem*, 30(6):445–451, 2006.
71. X.M. Zhao, R.S. Wang, L. Chen, and K. Aihara. Uncovering signal transduction networks from high-throughput data by integer linear programming. *Nucleic Acids Res*, 36(9):e48, 2008.

# 31

# HETEROGENEITY OF DIFFERENTIAL EXPRESSION IN CANCER STUDIES: ALGORITHMS AND METHODS

Radha Krishna Murthy Karuturi

## 31.1 INTRODUCTION

Cancer is an uncontrolled growth of abnormal cells evading apoptosis. It is a result of the failure of cell cycle control and apoptosis functions combined with an increased proliferation activity. Tumor formation and its progression to cancer causes major complex changes to the cells' maps of genome, transcriptome, and pathway regulation [10, 34, 35]. For example, the genomes of cancer cells, compared with that of normal cells, contain extra copies of some chromosomal segments as a result of duplication, translocation, and deletion of some regions of the genome [34, 35]. Erratic genomic changes and cell cycle regulation lead to different pathways being activated and different sets of genes coming into play in defining the state of the cells. The changes depend on many factors related to the patient such as the genetic makeup and the stage of the tumor. For example, *Grade*3 tumors will have many more structural variations (amplifications and deletions) in their genomes and a stronger expression of proliferation associated genes compared with *Grade*1 tumors. Estrogen receptor (ER)+ tumor progression may be facilitated by different

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

transcription factors and pathways compared with ER– tumors. Similarly, $p53+$ tumors grow potentially because of deregulation of the pathways downstream of p53, whereas the $p53-$ tumors grow because of the deregulation of the different pathways [4]. DNA mythelation also has been observed frequently causing some important apoptotic genes silenced leading to aberrant cell growth. All these factors lead to different transcriptional programs in different patients suffering from the same cancer. These transcriptional programs need to be analyzed appropriately to understand the underlying biology of the tumor origin and growth.

High-throughput technologies such as microarrays [27, 28] have made the simultaneous profiling of mRNA (expression) levels of all genes in the cells of a tumor sample feasible. The technological shift from measuring single gene expression to all-genes' expression has facilitated a major shift in identifying the genes involved in the origin and progression of tumors leading to a comprehensive understanding of the involvement of different pathways in cancers. The available data may be of cell lines or tumors with control samples or tumors without control samples. The number of samples also may range from the tens to hundreds.

The diversity of the sample composition of the datasets and genome-wide profiling of mRNA levels combined with diverse mechanisms of cancer progression requires orthogonal analytical approaches to extract important information about the relevant changes to the maps of genome, transcriptome, and the regulation of various pathways.

A variety of approaches have been proposed in the literature to elicit the biology of tumor formation and progression from different types of gene expression data. They analyze each gene separately or gene sets. They include assessing the differential expression in mean, differential variance of expression, differential expression in genomic localization, differential expression using gene–gene interactome, and differential coexpression. The chapter is aimed at describing these approaches to identify differential expression in tumors along with a brief description of a few appropriate methods available, but not meant to be a comprehensive survey of the methods, for each approach.

## 31.2 NOTATIONS

Throughout the chapter, we will be dealing with microarray data measuring the expression of $M$ genes among $N$ tissue (tumor or normal) samples. Following are the notations and the definitions used to describe the methods and the algorithms for a variety of differential expression analyses.

| | |
|---|---|
| $K$ | Total number of sample groups in the study (*i.e.*, $k \in \{1,2,\ldots,K\}$)<br>$>1$ for supervised analysis<br>$= 1$ for unsupervised analysis |
| $n_k$ | Number of samples in group $G_k = \{p_j \mathbin{/} \text{j} = 1, 2, \ldots, n_k\}$ |

$N$ Total number of samples in the study

$$= \sum_{k=1}^{K} n_k$$

$X_{ijk}$ Expression of gene $g_i$ in sample $p_j$ in group $G_k$
$X_{ij}$ is used instead if $K = 1$, mainly in unsupervised analyses

$m_{ik}$ Sample mean of expression of gene $g_i$ in group $G_k$ over all $n_k$ samples

$$= \frac{1}{n_k} \sum_{j=1}^{n_k} X_{ijk}$$

$m_i$ Sample mean of expression of gene $g_i$ over all $N$ samples in the study

$$= \frac{1}{N} \sum_{k=1}^{K} \sum_{j=1}^{n_k} X_{ijk}$$

$\sigma_{ik}^2$ Sample variance of expression of gene $g_i$ in group $G_k$ over all $n_k$ samples

$$= \frac{1}{n_k - 1} \sum_{j=1}^{n_k} (X_{ijk} - m_{ik})^2$$

$\sigma_i^2$ Sample pooled variance of expression of gene $g_i$ over all $K$ groups

$$= \frac{1}{N - K} \sum_{k=1}^{K} (n_k - 1)\sigma_{ik}^2$$

$R_{ijk}$ Rank of $X_{ijk}$ among all $N$ samples for a particular gene $g_i$

$R_{ik}$ Rank sum of all $n_k$ samples in group $G_k$ of gene $g_i = \sum_{j=1}^{n_k} R_{ijk}$

$\mathcal{N}(v/\mu,\sigma^2)$ Normally distributed random variable $v$ with mean $\mu$ and variance $\sigma^2 = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(v-\mu)^2}{2\sigma^2}}$

$\mathcal{X}^2(v/d)$ Chi-square distributed random variable $v$ with $d$ degrees of freedom $= \begin{cases} \frac{1}{2^{d/2}\Gamma(d/2)} v^{(d/2)-1} e^{-v/2} & for\ x > 0 \\ 0 & for\ x \le 0 \end{cases}$
where $\Gamma(.)$ is a gamma function

$\mathcal{F}(v/d_1, d_2)$ $F$-distributed random variable $v$ with $d_1$ and $d_2$ degrees of freedom

$$= \frac{1}{vB(d_1/2, d_2/2)} \sqrt{\frac{d_1^{d_1} d_2^{d_2} v^{d_1}}{(d_1 v + d_2)^{d_1 + d_2}}}$$ where $B(.)$ is a beta function

Med$_{ik}$ Median of the expression of gene $g_i$ in group $G_k$ over all $n_k$ samples

Med$_i$ Median of the expression of gene $g_i$ over all $N$ samples

| | |
|---|---|
| $\text{MAD}_{ik}$ | Median absolute deviation of the expression of gene $g_i$ in group $G_k$ over all $n_k$ samples = 1.4826 × Median $\{\lvert X_{ijk}-\text{Med}_{ik}\rvert / \lvert j = 1,2,\ldots\ n_k\}$ |
| $\text{MAD}_i$ | Median absolute deviation of the expression of gene $g_i$ over all $N$ samples |
| $p$-value | The probability that an observed statistic or its extreme can be a result of chance alone. Let the statistic $v$ follows a distribution with probability density function $f(v/\theta)$, and $\theta$ is the parameter vector of the distribution. Then the $p$-value of $v = \eta$ is $\int\limits_{\eta}^{\infty} f(v/\theta)dv$ |
| $Q1_{ik}, Q2_{ik}$<br>$Q3_{ik}, IQR_{ik}$ | First, second, and third quartiles of the expression of $g_i$ in $G_k$, respectively<br>$Q2_{ik} = \text{Med}_{ik}$; $\text{IQR}_{ik}$ is interquartile range $= Q3_{ik} - Q1_{ik}$ |
| DE | Differentially expressed |
| Non-DE | Nondifferentially expressed |
| Normality<br>Assumption | The random variable $x$ is assumed to follow a normal distribution with mean $\mu$ and variance $\sigma^2$ |
| $\cup$ | Set union |
| $\lvert D\rvert$ | Cardinality of the set $D$ (*i.e.*, number of elements in the set $D$) |
| $\lvert x\rvert$ | Absolute value of a real valued number $x$<br>$= x$ if $x \geq 0$<br>$-x$ if $x < 0$ |
| iid | independent and identically distributed |
| $U(x)$ | Step function<br>$= 1$ if $x > 0$<br>$0$ if $x \leq 0$ |
| $x \leftarrow y$ | The value of $x$ is replaced by that of $y$. $y$ may be function of $x$ itself. |

## 31.3 DIFFERENTIAL MEAN OF EXPRESSION

The traditional way of identifying genes relevant to a given cancer is by measuring the change of their mRNA levels in tumor tissues compared with the reference tissues and choosing the significantly changed genes, which are called differentially expressed. It has been of great focus even in the era of microarrays and sequencing, especially in the context of biomarker discovery. Several statistical procedures are available to identify differentially expressed genes (*e.g.*, $t$-test, Analysis of Variance [ANOVA], linear modeling, and their empirical Bayes versions significance analysis of microarrays [SAM] and linear models for microarray data [LIMMA] *etc.*)

The expression of a gene may be differentially influenced by different factors and their levels. Though the approaches used for a multifactor multilevel differential expression analysis are also applicable to two-level single factor analysis, well-tailored

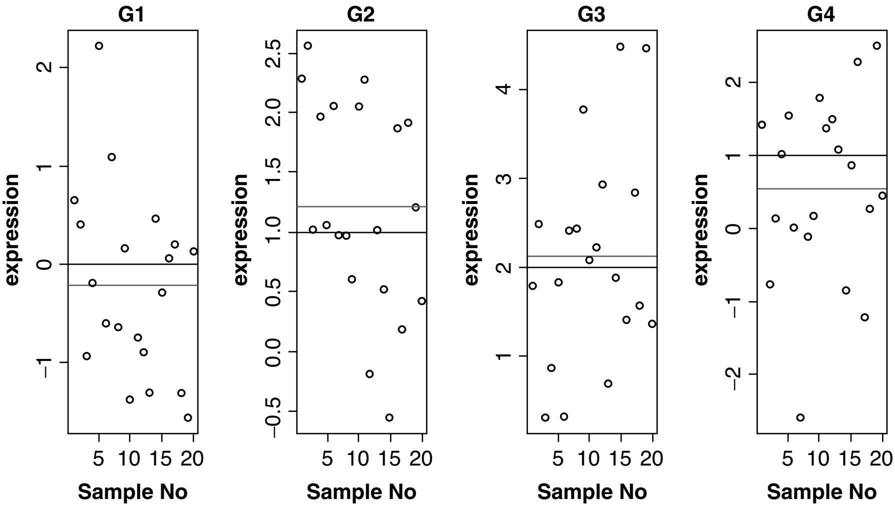


**Figure 31.1** Expression of a DE gene among four groups of samples ($G_1$, $G_2$, $G_3$, and $G_4$). Each circle is one sample, the black line indicates true expression, and the grey line indicates the estimated expression. Although $G2$ and G4 have same expression, their estimates are different. They are different from the remaining two groups. These differences among $G_1 \ldots G_4$ need to be tested.

methods that may yield better sensitivity and specificity are available for special cases. Hence, we will present separately the methods available for two-level single factor analysis, multilevel single factor analysis, and multifactor analysis.

### 31.3.1 Single Factor Differential Expression

In a single factor differential expression analysis, the factor is discrete or continuous valued. The factor could be ER status (ER+ vs. ER–, a two-level factor), $p53$ mutation status ($p53+$ vs. $p53$–, a two-level factor), histological grade of the tumor (*Grade*1, *Grade*2, and *Grade*3, a multilevel factor), treatment response (continuous valued in terms of dosage), or any other factor of interest. If the expression of a gene is significantly different in any level of the factor relative to the other levels, then the gene will be declared as differentially expressed (see Figure 31.1).

***31.3.1.1 Two-Level Single Factor Differential Expression Analysis.*** A two-level differential expression in mean can be handled by two-group differential expression analysis using the $t$-test [13] and the Mann–Whitney–Wilcoxon test [39]. Each level of the factor is treated as a group.

$T$-test measures the ratio ($T_i$) of the estimate of the differential mean of expression and its standard deviation.

$$T_i = \frac{m_{i2} - m_{i1}}{\sigma_{ie}} \sqrt{n_e} \sim t(T_i/n_2 + n_1 - a)$$

where $\sigma_{ie}$ is the standard deviation of the random variable $m_{i2} - m_{i1}$; it will be equal to the standard deviation of the tested group ($\sigma_{i2}$) if $m_{i1}$ is a constant reference (usually 0) or to the pooled variance ($\sigma_i^2$) if $m_{i1}$ is an estimate from a random group of $n_1$ samples. Similarly, $n_e$ is the effective number of samples used in the test; $n_e = n_2$ if $m_{i1}$ is a constant reference, and $1/n_e = 1/n_2 + 1/n_1$ if $m_{i1}$ is estimated from a reference group of size $n_1$. The statistic $T_i$, follows the central $t$-distribution with $n_2 + n_1 - a$ degrees of freedom, where $a = 1$ for a constant $m_{i1}$ and $a = 2$ for an estimated $m_{i1}$. $T$-test is mainly useful under normality assumption in both $G_1$ and $G_2$ with a smaller number of samples in each level.

The Mann–Whitney–Wilcoxon test is a nonparametric test; it is particularly powerful if the data does not obey the normality assumption and if $n_1$ and $n_2$ are large enough. The test is based on the distribution of the ranks of the samples between two groups instead of their values. The Mann–Whitney–Wilcoxon test statistic $z_i$ for gene $g_i$ is

$$z_i = \frac{\xi_i - m_{\xi_i}}{\sigma_{\xi_i}} \sim \mathcal{N}(z_i/0, 1)$$

where

$$\xi_i = \min\left(R_{i1} - \frac{n_1(n_1+1)}{2}, R_{i2} - \frac{n_2(n_2+1)}{2}\right)$$

$$m_{\xi_i} = \frac{n_1 n_2}{2}; \quad \sigma_{\xi_i} = \sqrt{\frac{n_1 n_2(n_1 + n_2 + 1)}{12}}$$

***31.3.1.2 Multilevel Single Factor Differential Expression Analysis in Mean.*** Although the $t$-test or Mann–Whitney–Wilcoxon test can be used for multilevel differential expression analysis by exploring all possible pairs, $K(K-1)/2$, of groups. It is not optimal because of the $K(K-1)/2$ tests needed for each gene. Therefore, multilevel differential expression in mean (as in grade analysis) can be handled by using ANOVA (or one-way ANOVA) [13] and the Kruskal–Wallis tests [13]. They test whether at least one pair of $\mu_{ik}$s (the true expression) is different using a single test.

ANOVA requires a gene's expression to follow a normality assumption in each group with the same underlying variance. The ANOVA statistic, $A_{Fi}$, measures the ratio of variance of the group means to the pooled variance of all groups put together as follows:

$$A_{Fi} = \frac{S_{1i}^2/(K-1)}{\sigma_i^2} \sim \mathcal{F}(A_{Fi}/K-1, N-K)$$

where

$$S_{1i}^2 = \sum_{k=1}^{K} n_k(m_{ik} - m_i)^2$$

$A_{Fi}$ follows the $F$ distribution with $(K-1,\ N-K)$ degrees of freedom under null hypothesis that $\mu_1 = \mu_2 = \mu_3 = \cdots = \mu_K$ (*i.e.*, expected value of $S_{1i}^2$, $E(S_{1i}^2), = 0)$.

The Kruskal–Wallis test is a nonparametric test for multigroup differential expression analysis. Similar to the Mann–Whitney–Wilcoxon test, the Kruskal–Wallis test uses ranks to test the null hypothesis. The ties are resolved by taking the mean of the range of the ranks of the tied values. The Kruskal–Wallis test statistic, $H_i$, for $g_i$ is

$$H_i = \left\{ \frac{12}{N(N+1)} \sum_{k=1}^{K} \frac{R_{ik}^2}{n_k} \right\} - 3(N+1) \sim \mathscr{X}^2(H_i/K-1)$$

$H_i$ follows $\mathscr{X}^2$-distribution with $K-1$ degrees of freedom.

### 31.3.2 Multifactor Differential Expression

Tumor samples can be represented by a vector of attributes such as ER status, grade, *p*53 mutation status, and treatments received. A simple two-group analysis on one factor may not give complete picture of the effects of various factors because of their confounding and interaction. For example, as in Figure 31.2, let the tumors

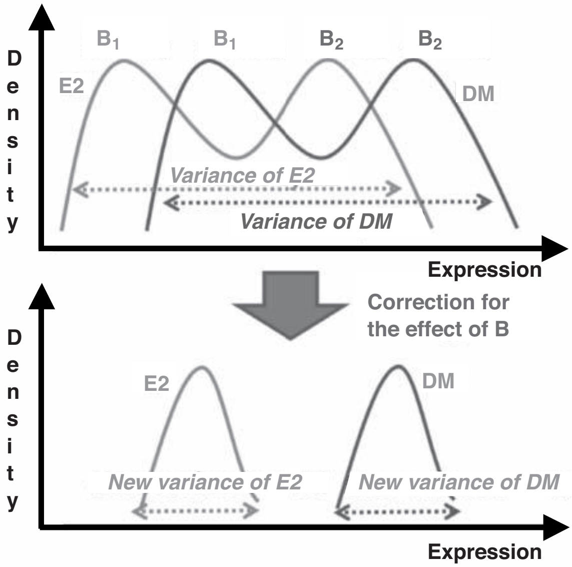


**Figure 31.2** Illustration of the additive effects of two factors on gene expression. The top and bottom figures show distributions of its expression in each treatment (*E*2 and *DM*) before and after correcting for the effects of the factor *B*. Adjustment for *B* gives much lower variance in both treatments, which gives better power to identify differential expression between treatments.

belong to two levels of factor $B$ ($B_1$ and $B_2$) in each level of the treatment ($E2$ and $DM$) received. The top figure shows the distribution of the expression (density plots) of a gene for two treatments; it can be seen easily that the variations of measurements are large from the influence of factor $B$. The bottom figure shows the same measurements, but with the adjustment for the effects of $B$, there is much lower variation leading to a potential increase in the power of the tests.

To deal with such a complex data, a meaningful analysis would involve all factors together (a multivariate approach). Linear models and generalized linear models [40] are useful to decipher the individual effects of the tumor attributes on the change in the expression levels of a gene. Note that all samples here are treated to belong to one group and the sample attributes (*e.g.*, ER status, $p53$ status, PR status, *etc.*) are treated as factors in the model.

For example, Miller *et al.* [4] used a linear model to identify $p53$ mutation specific changes in the expression of nearly 30,000 genes using a cohort of >250 tumors. The expression of a gene is modeled as a combination of tumor grade, ER status, and $p53$ mutation status as shown in the following equation.

$$X_{ij} \sim G_j + ER_j + p53_j + \varepsilon_{ij}$$

where $\mathcal{N}(\varepsilon_{ij}/0, s_i^2)$, $G_j$(grade) $\in$ *Grade*1, *Grade*2, *Grade*3$\}$, ER$_j$ (ER status) $\in$ $\{ER+, ER-\}$, and $p53_j$ (p53 status) $\in$ $\{p53+, p53-\}$.

$\varepsilon_{ij}$ is error and it is assumed to be iid and normally distributed with mean 0 and variance $s_i^2$. The least-squares minimization procedure is used to estimate the coefficients of the factors. The statistical significance of the estimated parameters of the linear model may be evaluated using the ANOVA procedure. The genes were ranked by the coefficient of the factor $p53$ status, and the top genes were selected to have been affected by $p53$ mutation specifically.

The multivariate approach is also useful to distinguish microarray batch effects from the actual factor effects. To achieve it, batch is used as a factor in the linear model as in COMBAT [37]. One may use generalized linear models if the errors are not distributed normally but belong to one of the members of the exponential family of distributions.

### 31.3.3 Empirical Bayes Extension

All of the above procedures were extended to include empirical bayes analysis. The error variance has been assumed to follow an inverse $\mathcal{X}^2$ distribution with mean $s_o^2$. The test statistics have been modified to accommodate the prior and are included in the packages such as SAM [6] and LIMMA [25]. They also have been included in the corresponding modified null distribution to estimate the statistical significance. Both SAM and LIMMA estimate the parameters of the prior from the data itself. The details have been omitted because of space limitation, interested readers can refer to the respective manuals for details.

## 31.4 DIFFERENTIAL VARIABILITY OF EXPRESSION

A gene deregulated in cancer does not necessarily demonstrate a significant unidirectional change in its mRNA levels in the tumor samples group. But, the change may be different in different tumor samples as the maps of a genome, transcriptome, and proteome of a tumor may not be same as the other tumors. This leads to different control mechanisms, and the genes may be regulated tightly in normal tissues but loosely or nonregulated in tumor tissues. In other words, the tightness of the control of a gene's expression changes from one group of tissues to another, and a change in the average level of expression may not be significant. This observation has led to the idea of identifying genes with differential variability in their expression [15] (see Figure 31.3).

### 31.4.1 *F*-Test for Two-Group Differential Variability Analysis

Each gene $g_i$ can be tested for its differential variability using an $F$-test [13] on the equality of variances by testing $S_{F1i}$ and $S_{F2i}$ for two different alternative hypotheses $\sigma_{i1}^2 > \sigma_{i2}^2$ and $\sigma_{i2}^2 > \sigma_{i1}^2$, respectively.

$$\begin{aligned} S_{F1i} &= \sigma_{i1}^2/\sigma_{i2}^2 \quad &\sim \mathscr{F}(S_{F1i}/n_1 - 1, n_2 - 1) \\ S_{F2i} &= \sigma_{i2}^2/\sigma_{i1}^2 \quad &\sim \mathscr{F}(S_{F2i}/n_2 - 1, n_1 - 1) \end{aligned}$$

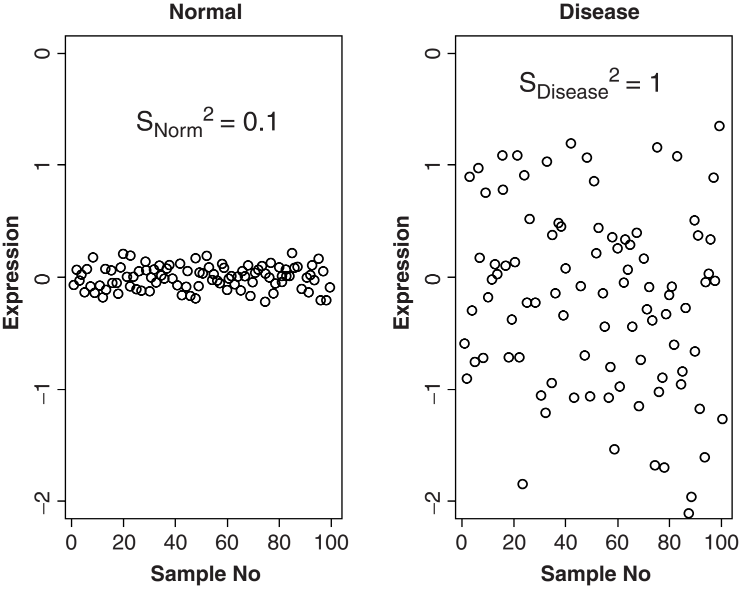


**Figure 31.3** Illustration of differential variability of expression of a gene between normal and disease tissues. The expression has a variance of 0.1 in normal tissues, whereas it is 1, 10-fold higher, in disease tissues. Note that the average expression level is 0 in both groups.

A gene is said to be differentially variable if either $S_{F1i}$ or $S_{F2i}$ is significant. They follow the $F$-distribution with $(n_1 - 1, n_2 - 1)$ and $(n_2 - 1, n_1 - 1)$ degrees of freedom respectively.

Ho *et al.* [15] proposed modifications to this procedure for robustness to outliers. One variation is removing outliers in both groups. For each gene $g_i$, in each group, all $X_{ijk}s$ falling outside the range $[Q1_i - 1.5IQR_i, Q3_i + 1.5IQR_i]$ are removed, and the $F$-test is conducted using the remaining $X_{ijk}s$. The other approach is to replace $\sigma_{ik}$ by $\text{MAD}_{ik}$ that is, $S_{F1i}$ and $S_{F2i}$ will change to

$$\begin{aligned} S_{F1i} &= \text{MAD}_{i1}^2/\text{MAD}_{i2}^2 \quad &\sim \mathcal{F}(S_{F1i}/n_1 - 1, n_2 - 1) \\ S_{F2i} &= \text{MAD}_{i2}^2/\text{MAD}_{i1}^2 \quad &\sim \mathcal{F}(S_{F2i}/n_2 - 1, n_1 - 1) \end{aligned}$$

### 31.4.2 Bartlett's and Levene's Tests for Multigroup Differential Variability Analysis

Because of the multiple testing involved, similar to the use of the $t$-test for multigroup differential expression testing, the $F$-test also is not recommended for multigroup differential variability analysis. Several tests, which do not require multiple testing, have been proposed in the statistics literature to test for differential variability among multiple groups. They test whether any two of the $K$ groups have differential variance using a single test. Two of the most well-known tests are from Bartlett and Levene.

Bartlett's test [13, 39] assumes normality of the data in each group. It is a modification of the corresponding likelihood ratio test designed to make the approximation to the $\mathcal{X}^2$ distribution better. Bartlett's differential variability test statistic for $g_i$, $V_i^B$ is

$$V_i^B = \frac{(N-K)\ln(\sigma_i^2) - \sum_{k=1}^{K} (n_k - 1)\ln(\sigma_{ik}^2)}{1 + \dfrac{1}{3(K-1)}\left(\sum_{k=1}^{K}\left(\dfrac{1}{n_k - 1}\right) - \dfrac{1}{N-K}\right)} \sim \mathcal{X}^2(V_i^B/K - 1)$$

The Bartlett's test statistic $V_i^B$ follows a $\mathcal{X}^2$ distribution with $K - 1$ degrees of freedom.

Levene's test [39] does not require a normality assumption of the underlying data unlike Bartlett's test. Levene's test measures a ratio $(V_i^L)$ for $g_i$ of the variance of the average absolute deviations to the pooled variance of the absolute deviations from the respective group means. It also can be described as conducting one-way ANOVA as in Section 31.3.1 on the absolute deviations $Z_{ijk} = |X_{ijk} - m_{ik}|$, which can be written as follows:

$$V_i^L = \frac{1}{K-1}\frac{\sum\limits_{k=1}^{K} n_k\,(m_{ik}(Z) - m_i(Z))^2}{\sigma_i^2(Z)} \sim \mathcal{F}(V_i^L/K - 1, N - K)$$

where $m_{ik}(Z)$, $m_i(Z)$, and $\sigma_i^2(Z)$ are the definitions similar to that of $m_{ik}$, $m_i$, and $\sigma_i^2$ but on $Z$ instead of $X$. $V_i^L$ follows the $\mathcal{F}$ distribution with $(K - 1, N - k)$ degrees of freedom. Levene's test can be more robust against outliers if we used the improvement shown in the Brown–Forsythe test [39]. It is different from Leven's test in the definition of $Z_{ijk}$, which is modified as $Z_{ijk} = |X_{ijk} - \text{Med}_{ik}|$. Improvements proposed by Ho *et al.* for a two-group case can be employed in Bartlett's and Levene's tests as well.

## 31.5 DIFFERENTIAL EXPRESSION IN COMPENDIUM OF TUMORS

The heterogeneity of tumors is of great interest in cancer studies. The knowledge of genetic differences that lead to the heterogeneity of tumors can be highly useful in the prognosis and treatment. To elicit such knowledge, we need to profile the gene expression in a compendium of tumors obtained through clinical trials or retrospective studies. This type of data lacks in replicates and control samples. Another important aspect of such data is that the number of subgroups defined by different genes could be different, and it is interesting to a researcher to discover all possible subgroups of the tumors. Unsupervised analysis or class discovery is preferred, as it requires discovery of inherent groups defined by different genes. The distribution of samples into the subgroups also changes from one gene to another gene, and it may be on either of the extremes—equidistribution of samples versus a skewed distribution of samples among the groups. There is no universal methodology to identify such genes in a compendium of profiles in an unsupervised analysis framework. Therefore, the genes that can delineate various subgroups of tumors are identified by different statistical techniques. We present three important techniques to identify genes that may help in class discovery. The fundamental idea is to test for nonnormality of the expression of a gene under Gaussian noise assumption [9, 16].

### 31.5.1 Gaussian Mixture Model (GMM) for Finite Levels of Expression

As finite discrete levels of expression of a gene and Gaussian distributed noise are assumed, we can model each gene's expression data by Gaussian mixture model (GMM) [12]. Gaussian Mixture Modeling approximates the distribution of a given data by a convex combination of a finite number of Gaussians. For example, the *pdf* $f_i(x)$ of $g_i$ can be approximated by a mixture of $K_i$ Gaussians with $\Pi_{ik}s$ as the mixing proportions:

$$f_i(x) \approx \sum_{k=1}^{K_i} \Pi_{ik}\mathcal{N}(x/\mu_{ik}, \sigma_{ik}^2); \sum_{k=1}^{K_i} \Pi_{ik} = 1; \Pi_{ik} \geq 0$$

The parameters $(\Pi_{ik}, \mu_{ik}, \sigma_{ik}^2)$ are estimated using the *expectation maximization* (EM) [12, 26] algorithm by minimizing the negative log likelihood $(O_{Ki})$ over the set of observations $\{X_{i1}, X_{i2} \dots X_{iN}\}$.

$$O_{K_i} = -\sum_{j=1}^{N} \log(f_i(X_{ij}))$$

The parameters of all $K_i$ Gaussians along with the mixing parameters are estimated by alternating between the expectation (E) step and the maximization (M) step.

**E step:**
In the E step, the association of $X_{ij}$ to the $k$th Gaussian ($\gamma_{ijk}$) is estimated

$$\gamma_{ijk} = p(k/X_{ij}) = \left\{\Pi_{ik}\mathcal{N}(X_{ij}/\mu_{ik}, \sigma_{ik}^2)\right\} \div \left\{\sum_{l=1}^{K_i} \Pi_{il}\mathcal{N}(X_{ij}/\mu_{il}, \sigma_{il}^2)\right\}$$

**M step:**
Based on the association ($\gamma_{ijk}$) estimated in the E step, the parameters of the $K_i$ Gaussians are estimated in the following sequence:

$$\Pi_{ik} = \frac{1}{N}\sum_{j=1}^{N}\gamma_{ijk}; \mu_{ik} = \frac{1}{N\Pi_{ik}}\sum_{j=1}^{N}\gamma_{ijk}X_{ij}; \sigma_{ik}^2 = \frac{1}{N\Pi_{ik}}\sum_{j=1}^{N}\gamma_{ijk}(X_{ij}-\mu_{ik})^2$$

The model complexity parameter $K_i$ is chosen using Akaike information criterion (AIC) [29] or Bayesian information criterion (BIC) [30] criterion commonly used in competitive nonnested model selection. The number of parameters in a $K_i$-GMM for $g_i$ with all $\Pi_{ik} > 0$ is $3K_i - 1$, as only $K_i - 1$ mixing parameters need to be estimated from the data. The AIC and BIC procedures are

$$\text{AIC:} \quad K_i^{\text{opt}} = \underset{K_i \in 1,..,N}{\arg\min}\, 6K_i - 2 + 2O_{K_i}$$

$$\text{BIC:} \quad K_i^{\text{opt}} = \underset{K_i \in 1,..,N}{\arg\min} (3K_i - 1)\ln(N) + 2O_{K_i}$$

Once the model is estimated using $K_i^{\text{opt}}$, the data can be discretized by defining boundaries ($b_{ik}s$) based on minimal error criterion between two consecutive Gaussians. The boundary $b_{ik}$ between $k$th and $(k+1)$th Gaussians ($\mu_{ik} \neq \mu_{i(k+1)}$) for a gene $g_i$ is obtained by solving the following equation: $\Pi_{ik}\mathcal{N}(x/\mu_{ik}, \sigma_{ik}^2) - \Pi_{i(k+1)}\mathcal{N}(x/\mu_{i(k+1)}, \sigma_{i(k+1)}^2) = 0$

The number $K_i^{\text{opt}}$ can be different for different genes (*i.e.*, the tissues may be divided into a different number of subgroups based on the gene under consideration). All genes whose $K_i^{\text{opt}} = 1$ will be considered as nondifferentially expressed genes as they fail to separate the samples into more than one group under Gaussian noise assumption.

### 31.5.2 Outlier Detection Strategy

If the number of tumors in which a gene is differentially expressed in a compendium of tumors is small (*i.e.*, <10%) then the GMM idea may not work well, especially with fewer samples, and qualify it to be a non-DE gene. Such a situation can be handled by the outlier detection strategy under Gaussian noise assumption. Although several methods have been available in the statistics literature to identify outliers, the most commonly used method is the $m_i \pm 3\sigma_i$ rule:

Upper outliers ($D_{Ui}$) and lower outliers ($D_{Li}$) for $g_i$ are identified using the following rules, see Figure 31.4:

$$D_{Ui} = \left\{X_{ij}/X_{ij} \geq m_i + r\sigma_i\right\}; \quad D_{Li} = \left\{X_{ij}/X_{ij} \leq m_i - r\sigma_i\right\}; \quad r > 0,$$
$$D_i = D_{Ui} \cup D_{Li}$$

Robust versions of these rules also have been adopted as one may incorrectly estimate the mean of the underlying reference distribution (shown by the peak in Figure 31.4) from a skewed overexpression or repression and also overestimate the variance of the underlying reference distribution owing to the presence of outliers. We can replace the mean (m$_i$) by the median and standard deviation ($\sigma_i$) by MAD$_i$, which changes these rules to

$$D_{Ui} = \left\{X_{ij}/X_{ij} \geq \text{Med}_i + r\text{MAD}_i\right\}; D_{Li} = \left\{X_{ij}/X_{ij} \leq \text{Med}_i - r\text{MAD}_i\right\}; r > 0$$
$$D_i = D_{Ui} \cup D_{Li}$$

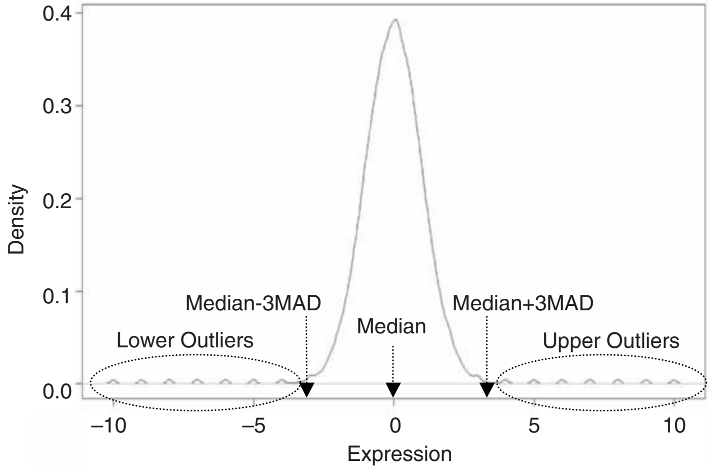


**Figure 31.4** Outlier detection strategy. For a given expression distribution, sample median and MAD are calculated, and the samples falling outside the range [Median − 3MAD, Median + 3MAD] are declared to be outliers in which the gene is differentially expressed.

Finally, the genes with at least $|D_i|/N > q \in [0,1]$ of outliers will be considered to be expressed differentially.

### 31.5.3 Kurtosis Excess

Some differentially expressed genes may not be identified if neither their expression is distinct enough between the different subgroups of tumors they define nor their distribution fits the outlier strategy. To test for differential expression in such circumstances, one may use Kurtosis excess, or simply Kurtosis, as a measure of differential expression [16, 17, 18]. Kurtosis measures the peakedness of the probability distribution of a gene's expression compared with its tails. For a normally distributed data, it is 0, and it will be <0 for fat-tailed distributions such as uniform distribution and mixture distributions.

Kurtosis [31] is defined as the ratio of the fourth cumulant ($C_4$) to the square of the second cumulant ($C_2$). It is equal to the ratio of the fourth central moment ($\lambda_4$) to the square of the second central moment (*i.e.*, $\sigma^2$) of the probability distribution minus three.

$$\text{Kurt}(X_i) = \frac{C_4(X_i)}{C_2^2(X_i)} = \frac{\lambda_4(X_i)}{\sigma_i^4(X_i)} - 3$$

Given the expression of a gene in a set of samples representing the whole population of patients of a certain cancer, the unbiased estimation of the kurtosis of gene expression is given by the ratio of unbiased estimators of the fourth and second cumulants as in the following equation:

$$\text{Kurt}(X_i) = \frac{N-1}{(N-2)(N-3)}\left((N+1)\frac{P_4(X_i)}{P_2^2(X_i)} - 3(N-1)\right)$$

where $P_4(X_i)$, $P_2(X_i) = \sigma_i^2$ are the fourth and second sample central moments, respectively, of $g_i$.

A gene is qualified to be expressed differentially if its kurtosis is sufficiently <0 under Gaussian noise assumption. The panel of plots in Figure 31.5 shows four different distributions. The first one is for a non-DE gene as its Kurtosis = 0, whereas the remaining represent DE genes with the varying number of divisions they create for the samples whose Kurtosis <0.

The test to check whether $\text{Kurt}(X) < 0$ (*i.e.*, whether the gene with expression sample $X_i$ is differentially expressed), we can use Fisher's cumulant test [13] by measuring $T_i^{\text{Kurt}}$, which follows the standard normal distribution, and a one-tail test is sufficient for our purpose as we are interested only in one alternative hypothesis $\text{Kurt}(X_i) < 0$.

$$T_i^{\text{Kurt}} = \text{Kurt}(X_i)(\mathscr{N}/24)^{0.5} \sim \mathscr{N}(0, 1)$$

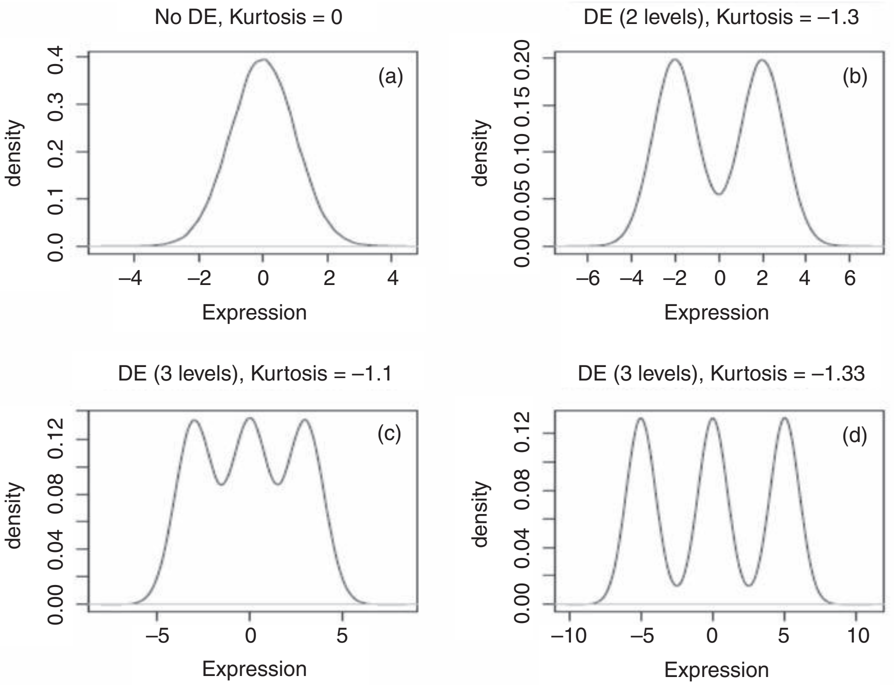


**Figure 31.5** Nonnormality and Kurtosis excess. The panel of plots shows expression distributions of four different genes. (a) Distribution of a non-DE gene's expression and its Kurtosis excess is 0. (b) Distribution of expression of a differentially expressed gene that divides samples into two groups as seen by two modes; its kurtosis excess is –1.3. Similar observations follow for the remaining two genes in plots (c) and (d).

Prior to the application of GMM, Kurtosis can be used to decide whether to fit GMM to speed up the modeling and identification steps by reducing the number of genes to be evaluated.

## 31.6 DIFFERENTIAL EXPRESSION BY CHROMOSOMAL ABERRATIONS: THE LOCAL PROPERTIES

Amplifications and deletions of chromosomal segments are common in tumors. They reflect in the transcriptome as induction and repression of the respective genes. As they are localized on chromosomes, the genes in such segments are expected to be expressed or repressed coordinately. This localized coordinated change of expression is critical to identify them from expression data.

Several researchers [7, 21, 36] have shown such a concordance between genomic amplifications and coordinated localized overexpression belonged to the amplified loci. For example, Pollack *et al.* [21] have shown it on breast cancer (see Figure 31.6). The heat maps of expression and copy number variations for whole Chr17

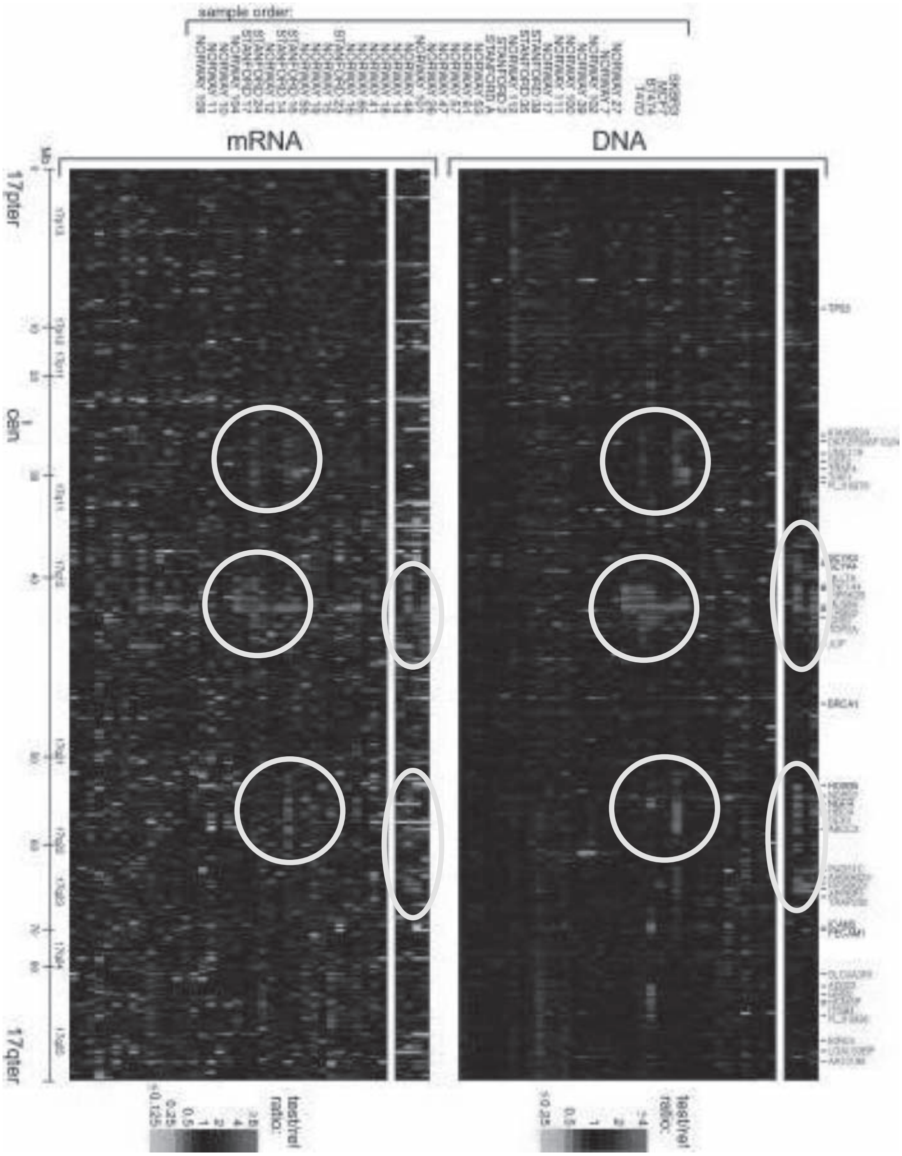


**Figure 31.6** Demonstration of the relationship between genomic amplification and localized coordinated induction of expression of the genes in the respective region. Shown are the heatmaps of expression (left) and copy number variation (CNV) (right) data from chr17. Probes are ordered by their genomic location for both expression as well as CNV heatmaps. The order of tumors is also the same in both heatmaps. Three regions can be observed with amplification, and the genes in them are expressed coordinately. (Pollack et al. [21]).

with the genes arranged by their chromosomal position and the samples were in the same order in both heat maps. It shows a correlation between overexpression and amplification, and it also shows the coordinated changes on proximally located genes in a subset of samples.

The genomic amplification or deletion maps are, in general, tumor specific. But, several regions of the map may be recurrent in several tumors which are of biological significance. The recurrence frequency of amplifications/deletions is usually small (<50%) and it depends on the locus and tumor. For example, amplification of a 1Mb region on 17q12 encompassing 30 genes including erythroblastic leukemia viral oncogene homolog 2 (ERBB2) recurs at a frequency of 20–30% in breast tumors [7] and it is absent in liver tumors [23]. Whereas, amplification of 20$q$13 occurs in 12% of primary breast tumors and in 11–39% of liver metastases of colorectal cancer. The frequency also changes with the stage of the tumor [24]. The loci of interest are the recurrently amplified/deleted ones. The challenge to identify them is a result of their low frequency of recurrence, below 30% typically, and the length of the amplified segment may change from tumor to tumor for a given locus. Moreover, the data may not include control samples or may contain only a few samples for the analysis as in the case of individual cell lines.

An appropriate choice of the methodology will be made based on the type of data. The general principle followed by any algorithm is sliding window analysis (see Figure 31.7). The genes are arranged in their chromosomal order, and each chromosome is analyzed separately. Overexpression/repression analysis is carried

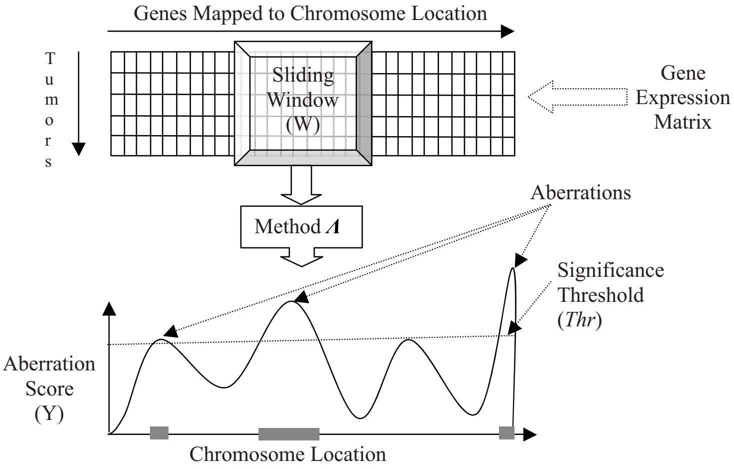


**Figure 31.7** Illustration of the generic approach to identify genomic aberrations from expression data. An expression matrix is generated for each chromosome. Genes are ordered by their location on the chromosome, and tumors can be in any arbitrary order. For each location on the chromosome, a submatrix of W genes with all tumors is derived, and the method $\Lambda$ is applied, which gives an aberration score Y for that location. The plot of Y versus the location shows peaks that are indicative of aberrations and the peaks above a threshold Thr are declared to be significant aberrations.

out on each window of genes of the chromosome. As a result, a plot of the coordinated changed expression-related measure, aberration score ($Y$), is obtained. The peaks above a certain significance threshold (Thr) in the plot indicate the aberrant genomic regions. The peaks in this plot show the respective aberrations. The higher the peak value $Y$, the higher the confidence of aberration around the locus. Finally, an aberration is reported at a peak with a range that covers a contiguous region with $Y >$ Thr with the center at the peak.

*False discovery rates* (FDR) are used to assess the statistical significance of the selected peaks by estimating the proportion of false positives from the total number of significant windows. The number of false positives at a given Thr is estimated by the permutation procedure. The data is randomized for their genomic location, and the same procedure is applied. The false discovery rate of windows in the rejection region (>Thr) is defined as FDR $= L^{\mathrm{rnd}}/L$, where $L$ is the number of aberrant regions in the actual data scan and $L^{\mathrm{rnd}}$ is the number of regions falsely called aberrant in the permuted data scan.

The following discussion on various methods ($\Lambda$) refers only to amplifications. It is also applicable to deletions, as the detection methodologies are the same for both amplifications and deletions.

### 31.6.1 Wavelet Variance Scanning (WAVES) for Single-Sample Analysis

Wavelet analysis [32] converts a spatial or temporal domain data into its frequency domain data using so-called wavelet basis functions. A wavelet function $\psi(x/u, s)$ is a 0-mean function with position parameter $u$ and scale parameter $s$:

$$\psi(x/u, s) = \frac{1}{\sqrt{s}}\psi\left(\frac{x-u}{s}\right)$$

$$\int_{-\infty}^{+\infty} \psi(x/u, s)dx = 0$$

The wavelet coefficient of $f(x)$,$Wf(u$,$s)$, at scale $s$ and position $u$ is computed by correlating $f(x)$ with $\psi(x/u$,$s)$ as follows:

$$Wf(u, s) = \int_{-\infty}^{+\infty} f(x)\psi^{*}(x/u, s)dx$$

where $\psi^{*}(x/u$,$s)$ is the complex conjugate of $\psi(x/u$,$s)$. Wavelet coefficients are obtained by varying the wavelet scale $s$ at each position of $u$. Wavelet analysis is unique in the sense that different scale parameters are used at every location to deal with the accuracy-resolution dilemma. Aggarwal *et al.* [20] have employed wavelet analysis to find coordinately expressed localized genes in several cell lines using Morlet wavelets, which are sine waves modulated by Gaussian curves.

$$\psi(x/u, s) = \pi^{\frac{-1}{4}} e^{i6(x-u)/s+(x-u)^2/2s^2}$$

The wavelet coefficients at different scales are estimated in each window $(W)$. To estimate the continuous wavelet transform, the scales are chosen to be 1, 2, 4, 8, 16, and 32 with four logarithmic subdivisions within each interval of two consecutive scales, resulting in 21 scales. The aberration score *Yu* at position $u$ is defined as follows:

$$Yu = \sum_{i=1}^{21} \frac{|Wf(u, s_i)|^2}{s_i}$$

The underlying fundamental is that no localized expression results in near-zero wavelet coefficients irrespective of the scaling (*i.e.*, near-zero *Yu*), whereas localized coordinated expression results in nonzero wavelet coefficients for some scales leading to positive *Yu*.

### 31.6.2 Local Singular Value Decomposition (LSVD) for Compendium of Tumors

The problem of identifying localized coexpression, in a compendium of tumors can be posed as a localized biclustering problem [7]. The expression data was discretized using the following outlier strategy: $X_{ij} \leftarrow 1$ if $X_{ij} \in D_{Ui}$, $X_{ij} \leftarrow -1$ if $X_{ij} \in D_{Li}$, and $X_{ij} \leftarrow 0$ otherwise. To detect amplifications, $X_{ij} \leftarrow 0$ if $X_{ij} < 0$ and vice versa. The biclustering problem, especially for a single bicluster, can be solved using *singular value decomposition* (SVD) [33].

SVD of a given $W \times N$ matrix $Ap_c$ decomposes it into a product of three matrices $Up_c$, $\Sigma p_c$, and $Vp_c$ (*i.e.*, $Ap_c = Up_c \times \Sigma p_c \times Vp_c^T$). $Up_c$ and $Vp_c$ are $W \times W$ and $N \times N$ matrices, respectively, whereas $\Sigma p_c$ is a $W \times N$ diagonal matrix. The column vectors of $Up_c$ and $Vp_c$ are the eigen vectors of $Ap_c Ap_c^T$ and $Ap_c^T Ap_c$, respectively whereas the diagonal elements of $\Sigma p_c$ are the respective eigen values. The largest eigen value is called the *principal eigen value* (denoted by $\lambda p_c$), and the respective eigen vectors are called the *principal eigen vectors*. The *eigen weight of tumor* $T_j$ from $Ap_c$, is denoted by $Tp_{cj}$, and it is defined as the absolute of the $j$th component of the *principal eigen vector* of the matrix $Ap_c^T Ap_c$. Similarly, the *eigen weight* of a gene $g_i$ from $Ap_c$ is denoted by $Gp_{ci}$ and is the absolute of the $i$th component of the principal eigen vector of the matrix $Ap_c Ap_c^T$. The absolute of the *principal eigen value* $\lambda p_c$ is the measure of the strength of the bicluster.

For example, the gene expression matrix on the left of the Figure 31.8 is analyzed with SVD, and reordering the tumors in the descending order of their eigen weights (from left to right) gives the matrix shown on the right. The shaded region is the

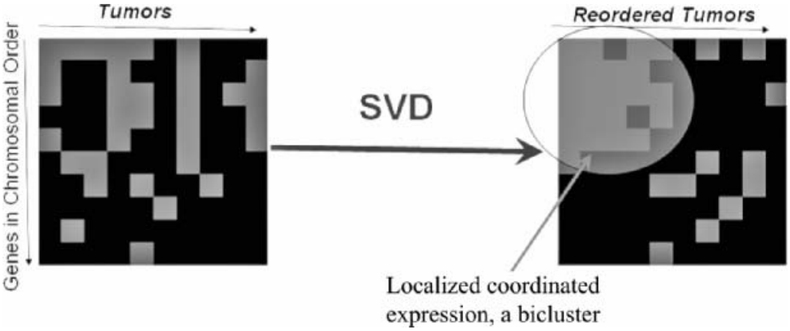

Tumors
Genes in Chromosomal Order
SVD
Reordered Tumors
Localized coordinated expression, a bicluster

**Figure 31.8** Illustration of the application of SVD to identify a bicluster from an expression matrix (the left heatmap). Application of SVD and reordering the tumors by their eigen weights (obtained from the principal eigen vector) reveals the bicluster in the heat map on right (the shaded region).

bicluster, which is a localized coordinated overexpression of a few genes on a few samples.

To improve the contrast between strong biclusters and weak or random biclusters, SVD on $(Ap_cAp_c^T)^3$ is computed to find $\lambda p_c$. The eigen weights of tumors and genes are obtained from the principal eigen vectors of $(Ap_c^TAp_c)^4$ and $(Ap_cAp_c^T)^4$, respectively. Now, $\lambda p_c$ is taken to be the ratio of the fourth root of the principal eigen value of $(Ap_cAp_c^T)^4$ to $N^4$; the eigen weights of genes and tumors are the fourth root of the respective components of the principal eigen vectors of $(Ap_cAp_c^T)^4/N^4$ and $(Ap_c^TAp_c)^4/N^4$, respectively.

As $\lambda p_c$ denotes the level of bicluster, which in turn represents the level of the localized recurrent coordinated overexpression of the genes in the window of W at position $p_c$, the aberration score $Yp_c = \lambda p_c$, and the principal eigen value resulted in the application of the SVD on the corresponding $W \times N$ matrix $Ap_c$ at $p_c$, hence, the name Local SVD (LSVD) (*i.e.*, localized application of SVD in the chromosomal order).

### 31.6.3 Locally Adaptive Statistical Procedure (LAP) for Compendium of Tumors with Control Samples

Although LSVD can be applied for any compendium of tumors by ignoring the sample labels, it may be appropriate to exploit the sample label information to explore the localized coordinated expression of genes especially for $N < 20$. The locally adaptive statistical procedure (LAP) is specially designed for such a labeled data. The LAP procedure [14] converts the expression of genes on a compendium of tumors with control samples into a one-dimensional plot of differential expression scores ($d_i$ for $g_i$) obtained based on SAM analysis as in Section 31.3. Hence, LAP is applicable to any tumor data with two or more groups.

The $d_i$ statistics are sorted and smoothed over the chromosomal coordinate, obtaining for each locus a smoothed statistic, which is the aberration score *Yu*. The

LAP procedure employs smoothing based on a local variable bandwidth kernel estimator, the *lokerns function* [38]. Lokern comprises a function that automatically estimates the optimal bandwidths iteratively. Polynomial kernels and boundary kernels are used with a fast and stable updating algorithm for kernel smoothing.

## 31.7 DIFFERENTIAL EXPRESSION IN GENE INTERACTOME

The gene interactome is defined as a gene–gene correlation structure obtained from the expression data under consideration or under other biological knowledge. The structure is either a directed or undirected graph with each node representing a gene and edges representing interaction defined based on a response variable (*e.g.*, sample type, survival outcome, *etc*). Such a structure is useful to define differential expression of a gene in the context of the other genes. Several methods based on a variety of interactions have been proposed in the literature. Here, we present three approaches: (i) the friendly neighbors (FNs) algorithm based on multiplicative interaction, (ii) the GENERANK algorithm based on contributing interaction defined using Gene Ontology (GO) and pathway databases, and (iii) the top scoring pairs (TSP) method and its generalization weighted top scoring pairs method based on differential interaction.

### 31.7.1 Friendly Neighbors Algorithm: A Multiplicative Interactome

The friendly neighbors algorithm [1] generates a graph of gene–gene interactions based on a single group data. A pair of genes ($g_a$,$g_b$) are said to be interacting or correlated if $|S(g_a,g_b/G_k)| \geq \text{Thr} \in [0.5,1]$; $S(g_a,g_b/G_k)$ is Kendall's correlation between $g_a$ and $g_b$.

$$S(g_a,g_b/G_k) = \frac{1}{n_k}\sum_{j=1}^{n_k} U((X_{ajk} - m_{ak}) \times (X_{bjk} - m_{bk}))$$

Now the interactome in group $G_k$ is a set of coexpressed pairs, Int($G_k$)

$$\text{Int}(G_k) = \{(g_a,g_b)/|S(g_a,g_b/G_k)| \geq \text{Thr} \in [0.5, 1]\}$$

Based on this definition of interactome, each gene can be assigned a score of importance in $G_k$ as the number of pairs or interactions it participates in Int($G_k$). In other words, the statistic of importance (FNs statistic) of a gene is the number of the other genes coexpressed with it in the data. For example, as shown in Figure 31.9, $g_1$ has four neighbors within its vicinity defined in the similarity space and the similarity threshold Thr, but $g_2$ has only one neighbor indicating that $g_1$ is more likely to be a responsive gene compared with $g_2$. Note that the gene itself may not show any trend of up regulation or down regulation in $G_{k.}$

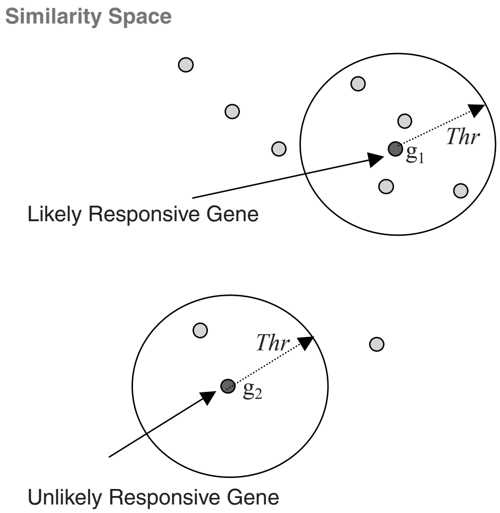


**Figure 31.9** Illustration of the friendly neighbors algorithm. In a predefined similarity space, a vicinity or neighborhood is defined for a given threshold Thr for each gene and the number of genes falling within the neighborhood, the FNs, is the score of the gene under consideration. The gene with more FNs is considered to be a likely responsive and important gene. For example, $g_1$ is more likely to be a responsive gene compared with $g_2$ in this illustration.

### 31.7.2 GeneRank: A Contributing Interactome

GENERANK [22] is a generalization of the friendly neighbors algorithm in which the interactome could be the correlation/multiplicative interactome or interactome defined by GO categories and pathways. Differential expression in mean is key to the analysis in GENERANK, as it assumes that there are some consistently differentially expressed (in mean) genes between groups. We can call this interactome a contributing interactome because it is defined based on certain predefined biological knowledge, and the interaction of genes is for contributing each other in defining differential expression. For example, as in the interactome in Figure 31.10, the DE

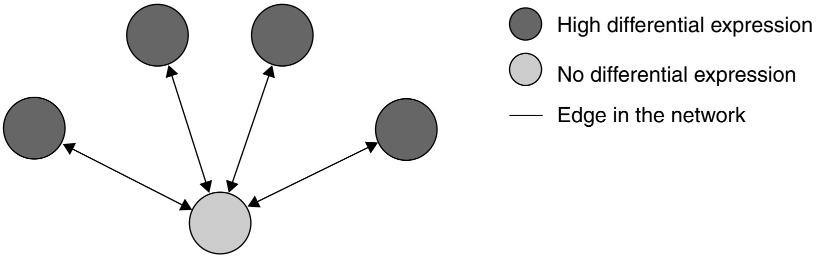


**Figure 31.10** Illustration of the GENERANK algorithm. Grey circle shows the gene whose differential expression is to be evaluated. The genes connected to it are shown in black circles, which are highly differentially expressed. Because of the interaction between them, the gene in Grey is declared to be differentially expressed.

of genes shown by the black circles contributes to defining the differential expression of the gene shown in the grey circle. The genes with connections to genes of a high differential expression carry greater significance than genes with connections to genes of a low differential expression.

Each gene is given an initial score of $(1-d)\mathrm{ex}_i$, where $\mathrm{ex}_i$ is the absolute value of the expression change for $\mathrm{g}_i$, and $d$ is the parameter $\in [0,1]$. Let $r_j^{[l]}$ denote the score of $\mathrm{g}_j$ after $l$th iteration and the initial score $r^{[0]} = ex/||ex||_1$, where $||ex||_1$ denotes the vector 1–norm of the vector of $ex_i$s. Then $r_j^{[l]}$, the convex combination of the gene's expression and the expression of the contributing genes in the $l$th iteration, is expressed as follows:

$$r_j^l = (1-d)ex_j + d\sum_{i=1}^{M} \frac{w_{ij} r_i^{l-1}}{\deg_i}; \quad 1 \le j \le M$$

where $w_{ij} = 1$ if $g_i$ is connected to $g_j$, and $w_{ij} = 0$ otherwise.

The extreme parameter values $d = 0$ and $d = 1$ represent ranking by pure expression level and pure degree respectively. Hence changing the value of $d$ allows interpolation between these two extremes.

### 31.7.3 Top Scoring Pairs (TSP): A Differential Interactome

Differential interactome is generated in the TSP algorithm [19], and it is defined based on the change in paired differential expression (DiffS) between genes $(g_a,g_b)$ from one group to the other group.

$$\mathrm{DiffS}(g_a,g_b/G_1,G_2) = \left| \frac{1}{n_1}\sum_{j=1}^{n_1} U\left(X_{aj1} - X_{bj1}\right) - \frac{1}{n_2}\sum_{j=1}^{n_2} U\left(X_{aj2} - X_{bj2}\right) \right|$$

The differential interactome DiffInt is

$$\mathrm{DiffInt}(G_1,G_2) = \{(g_a,g_b)/\mathit{DiffS}(g_a,g_b|G_1,G_2) > \mathit{Thr} \in [0,\ 1]\}$$

As shown in the equation, a differential interactome is defined for a pair of groups. Two genes are considered to be interacting if their pair-wise differences in one group are consistent in one direction compared with the other group. The DiffInt$(G_1,G_2)$ generates features to generate classifiers as illustrated in Figure 31.11. The expression of $g_1$ (black curve) and $g_2$ (grey curve) in groups $G_1$ and $G_2$ are shown. In $G_1$, the expression of $g_1$ is less than that of $g_2$ in eight of 10 samples (80%). In contrast, in $G_2$, $g_1$ has a lower expression than $g_2$ in only five out of 10 tumors (50%). Note that neither gene may have significant nonzero expression in any of the groups $G_1$ and $G_2$. But, their difference can serve as a useful feature for classification of the samples into groups $G_1$ and $G_2$.

Luo *et al.* [11] have generalized the DiffS $(g_a,g_b/G_1,G_2)$ score by taking into account the case of unequal $n_1$ and $n_2$. The modified DiffS$(g_a,g_b/G_1,G_2)$ uses

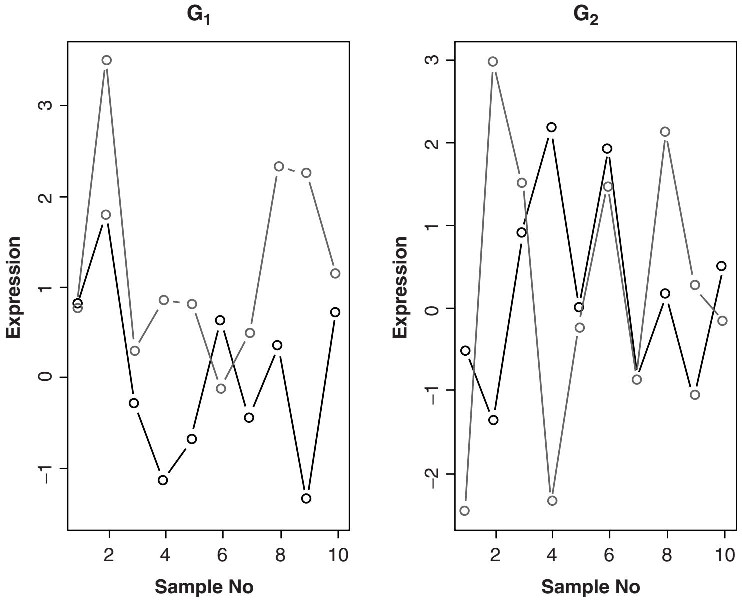


**Figure 31.11** Illustration of the differential interactome. The expression of two genes in two sample groups ($G_1$ and $G_2$) is shown by grey and black curves. The grey curve is above the black curve for 80% of the samples in $G_1$, whereas it is only 50% in $G_2$. Therefore, the difference between the two is a potential feature to distinguish the samples into groups $G_1$ and $G_2$ (*i.e.*, if $X_{aj1} - X_{bj1} < 0$, then $p_j$, likely belongs to $G_1$).

weights $\lambda_k n_k$ for each term, resulting in $\text{DiffS}^M(g_a,g_b/G_1,G_2,\lambda_1,\lambda_2)$, hence, the name weighted TSP (WTSP).

$$\text{DiffS}^M(g_a,g_b/G_1,G_2,\lambda_1,\lambda_2) = \left|\lambda_1\sum_{j=1}^{n_1}U\left(X_{aj1}-X_{bj1}\right)-\lambda_2\sum_{j=1}^{n_2}U\left(X_{aj2}-X_{bj2}\right)\right|$$

$\text{DiffS}^M$ minimizes the cost of misclassification. If $\lambda_k = 1$, then it minimizes the misclassification rate by the differential feature $X_{aj} - X_{bj}$. Moreover, $\text{DiffS}(g_a,g_b/G_1,G_2)$ can be derived from $\text{DiffS}^M(g_a,g_b/G_1,G_2,\lambda_1,\lambda_2)$ if $\lambda_k$ is chosen to be $1/n_k$ (*i.e.*, $\text{DiffS}^M(g_a,g_b/G_1,G_2,\lambda_1 = 1/n_1,\lambda_2 = 1/n_2) = \text{DiffS}(g_a,g_b/G_1,G_2)$).

## 31.8 DIFFERENTIAL COEXPRESSION: GLOBAL MULTIDIMENSIONAL INTERACTOME

Differential coexpression is a phenomenon observed when a set of genes are well controlled by a biological mechanism in one set of tissues such as normal, and the control is lost in the other set of tissues such as tumors. The gene set demonstrates

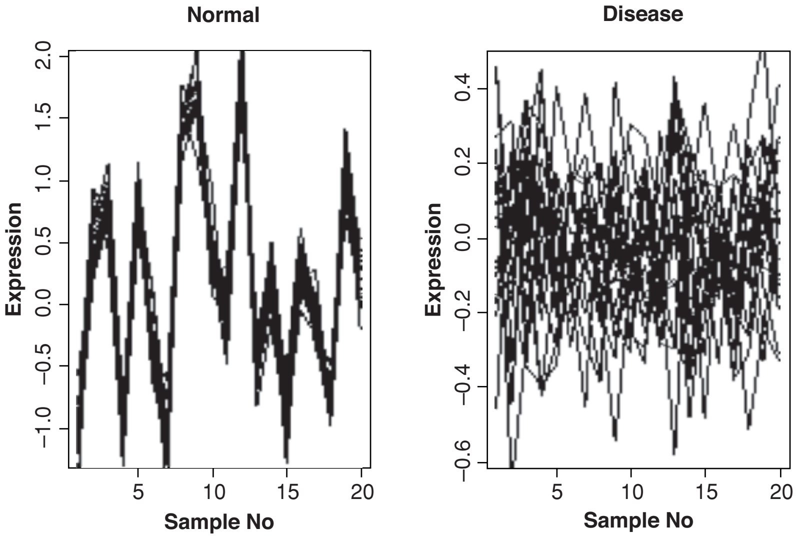


**Figure 31.12** Illustration of differential coexpression. Left: coexpression of a gene set in normal samples. Right: no coexpression of the same gene set in disease samples (*e.g.*, tumors). It can be noted that differential coexpression may not result in differential expression.

a high level of coexpression in the first set of tissues because of tight regulation and because of a relatively poor level of coexpression as a result of loss of regulation in the disease samples, (see Figure 31.12).

A variety of formulations have been proposed that differ in the conceptual and mathematical definition of differential coexpression. Three algorithms to identify differentially coexpressed gene sets are discussed here.

### 31.8.1 Kostka and Spang's Differential Coexpression Algorithm

Kostka and Spang [3] pioneered the idea of differential coexpression. The differential error of two linear models fit is the basis of quantifying differential coexpression, and a stochastic downhill search algorithm (we call it the KS algorithm) was proposed to identify significantly differentially coexpressed gene sets.

The formulation is for $K = 2$; the first group is for normal samples, and the second group is for diseased samples. The task is to find a subset of genes $I$ that have a high differential coexpression pattern from $M$ genes. To find such an $I$, the KS algorithm used the ratio of sum squared errors resulting from fitting linear models for both sets of $G_k$ $(k = 1, 2)$. Specifically, the linear model of expression on $I$ genes in sample group $G_k$ is expressed as follows:

$$X_{ijk} = \mu_k + \beta_{jk} + \tau_{ik} + \varepsilon_{ijk}$$

$$\varepsilon_{ijk} \sim \mathscr{N}(0, \sigma_{ik}^2);\ 1 \le i \le I,\ 1 \le j \le n_k \text{ and } 1 \le k \le 2$$

$X_{ijk}$ is modeled as a summation of the following factors: $\mu_k$ is the effect of group (overall effect) $G_k$; $\tau_{ik,}$ is the effect of gene $g_i$ in $G_k$; $\beta_{jk}$ is the effect of sample $p_{jk}$ in $G_k$; $\varepsilon_{ijk}$, an random error or residual of $g_i$ in $p_{jk}$. Based on this model, Kostka and Spang obtained the mean of the squared residuals for scoring a set of $I$ genes $E_k(D_{IxJk})$ (also referred as $E_k$) as follows:

$$E_k = \frac{1}{(I-1)(n_k-1)} \sum_{i=1,j=1}^{I,n_k} \left(X_{ijk} - \hat{\tau}_{ik} - \hat{\beta}_{jk} + \hat{\mu}_k\right)^2$$

$$\hat{\beta}_{jk} = \frac{1}{I}\sum_{i=1}^{I} X_{ijk}; \quad \hat{\tau}_{ik} = \frac{1}{n_k}\sum_{j=1}^{n_k} X_{ijk}$$

$$\hat{\mu}_k = \frac{1}{In_k}\sum_{i=1}^{I}\sum_{j=1}^{n_k} X_{ijk} = \frac{1}{n_k}\sum_{j=1}^{n_k}\hat{\beta}_{jk} = \frac{1}{I}\sum_{i=1}^{I}\hat{\tau}_{ik}$$

where $\hat{\tau}_{ik}$, $\hat{\beta}_{jk}$, $\hat{\mu}_k$ are the estimates of $\tau_{ik}$, $\beta_{jk}$, $-\mu_k$, respectively. A group of genes with a low score $E_k$ is highly expressed in group $G_k$. The KS statistic $S(I)$ is the ratio of $E_k$'s,

$$S(I) = E_1/E_2$$

The statistic evaluates how tightly the $I$ genes are correlated in one group compared with the other group. A set of genes with a distinguished low score $S(I)$ are coexpressed in group one but not in group two. But to take into account the effect of the size of $I$ on the statistical significance of $S(I)$, the $S(I)$ has been augmented with a regularization term including $|I|$ with a regularization parameter $\alpha$. With this addition, a set of differentially coexpressed genes can be obtained by minimizing the score $S(I)$ over $I$. The algorithm starts with an initial random set of $I$. The iterative stochastic downhill search procedure either eliminates a gene from $I$ or adds new gene to $I$. The gene g$_h$ to be added to $I$ provides locally a maximum improvement in the score $\Delta S(I) = S(I + g_h) - S(I)$. Similarly, the gene $g_g$ to be removed from $I$ also provides locally a maximum improvement in the score $\Delta S(I) = S(I - g_g) - S(I)$. The algorithm iterates until $I$ is not changeable. The recommended choice for $\alpha$ is 0.5, although it is arbitrary.

But, before application of the algorithm on a real data, Kostka and Spang have normalized the data as follows:

$$X_{ijk} \leftarrow \frac{X_{ijk} - m_{ik}}{\sigma_{ik}} \quad \text{(Eq35)}$$

Li and Karuturi [8] have shown that the squared inverse of the unregularized (*i.e.*, no $\alpha$ term) KS statistic $(E_1/E_2)^2$ on a normalized data (Eq35) follows doubly noncentral $F$-distribution.

$$\frac{J_2 - 1}{J_1 - 1}\frac{\sigma_{F1}^2}{\sigma_{F2}^2}\frac{1}{S^2(I)} \sim F\left(1, 1, \frac{\mu_{F2}^2(I-1)}{\sigma_{F2}^2/(J_2-1)}, \frac{\mu_{F1}^2(I-1)}{\sigma_{F1}^2/(J_1-1)}\right)$$

They have shown that optimal $\alpha$, $\alpha_{\text{opt}}$, can be estimated by maximizing the statistical significance of the set instead of the regularized KS statistic on the standard normalized data, that is,

$$\alpha_{\text{opt}} = \arg\max_{\alpha}\,(-\log(P_\alpha))$$

On a simulated data, $\alpha_{\text{opt}}$ varied for different choices of noise variance in the simulation as shown in the Figure 31.13. Low noise variance simulation allows for broad choices of $\alpha$, whereas the medium noise has its $\alpha_{\text{opt}} < 0.4$.

They also have proposed improving the performance of the algorithm by prefiltering the genes using the FNs algorithm on group 1 in which $I$ is coexpressed. Let $S_T$ denote the set of FNs statistics of all genes in the dataset for a given Thr, the lower bound of the rejection region of the FNs statistic may be chosen as

$$\text{median}(S_T) + \max(\text{MAD}(S_T), I^e/3)$$

where $I^e$ is the expected number of coexpressed genes in group 1. This rule will filter out 50 to ~100% of the noncontributing genes.

After having chosen genes with an FNs statistic above the threshold, the KS algorithm is applied on the selected set of genes instead of the entire set $M$. The

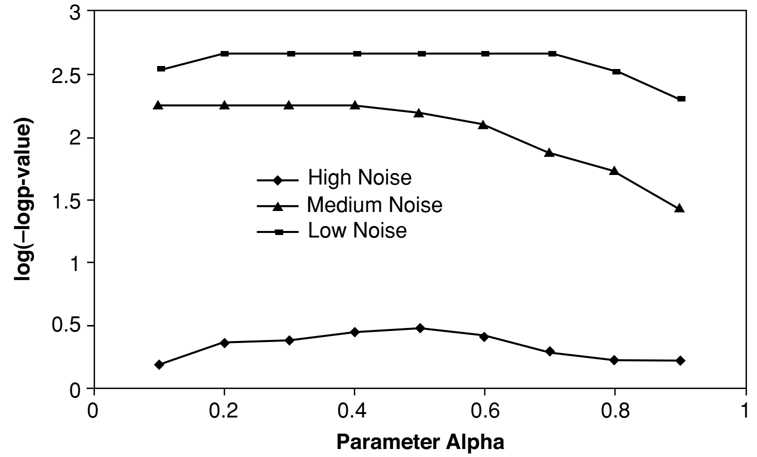


**Figure 31.13** Selection of $\alpha_{\text{opt}}$. The graph shows the log of –log of the $p$-value of the set obtained by the given $\alpha$. $\alpha_{\text{opt}}$ is chosen to be the one that gives the highest significance. The graph has three plots for different noise levels in the simulation.

application of the KS algorithm still is required because the gene set output by FNs may contain false positives, especially when $n_1$ is small or when noise variance is higher.

The objective optimal choices of Thr and $I^e$, $\text{Thr}_{\text{opt}}$ and $I^e_{\text{opt}}$, are made using the similar criterion as in case of $\alpha_{\text{opt}}$, that is,

$$\text{Thr}_{\text{opt}} = \arg\max_{T} \left(-\log(P_T)\right)$$

$$I^e_{\text{opt}} = \arg\max_{I^e} \left(-\log(P_{I^e})\right)$$

Because the choices of $\alpha$, $T$, and $I^e$ may not be chosen independently, we can select an optimal combination of $\alpha$, $T$, and $I^e$ by choosing the optimal triplet $(\alpha,T,I^e)_{\text{opt}}$.

### 31.8.2 Differential Expression Linked Differential Coexpression

In contrast to Kostka and Spang, Preito *et al.* [5] have proposed a formulation that especially takes into account the differential expression linked to differential coexpression, which allows for coordinated differential variability in $I$ (*i.e.*, no standardization unlike in Kostka and Spang's approach). Their procedure also is based on the same linear model formulation as used in Kostka and Spang's idea. The genes that do not show a minimum differential expression between the groups are filtered out, and the remaining genes were explored to elicit $I$ that minimizes $S(I)$. Another major difference is that Prieto *et al.*'s algorithm outputs a series of $I$s.

The basic idea of the algorithm, similar to Kostka and Spang's, is to allow a progressive selection of groups with an increasing score from a minimal initial value that may correspond to the nondifferential coexpression.

### 31.8.3 Differential Friendly Neighbors (DiffFNs)

*Differential friendly neighbors* (DiffFNs) [2], unlike the approaches proposed by both Kostka and Spang and Preito *et al.*, does not discount either the differential coexpression or the covariance of genes in the definition of a set for differential coexpression.

DiffFNs is an extension of the friendly neighbors algorithm to two-group data. The DiffFNs formulation defines a gene to be significant if it significantly gains or loses correlated genes (FNs) from one set of samples to the other set.

$X_{\text{ijk}}$ is discretized into one of the $\{-1, 1\}$ as follows:

$$X_{\text{ijk}} \leftarrow 2U\left(X_{\text{ijk}} - \frac{m_{i1} + m_{i2}}{2}\right) - 1$$

Then the similarity between two genes $g_a$ and $g_b$ in group $G_k$ is defined as follows:

$$\Phi_{\text{abk}} = \frac{1}{n_k}\sum_{j=1}^{n_k}\left(2U\left(X_{\text{ajk}} \times X_{\text{bjk}}\right) - 1\right)$$

$\Phi_{\text{abk}}$ measures the proportion of samples in which both genes have the same direction of expression from their overall mean $(m_{i1} + m_{i2})/2$ (*i.e.*, $\Phi_{\text{abk}} = 1$ if both genes have the same direction of expression [positive correlation], $\Phi_{\text{abk}} = -1$ if they have exactly the opposite direction of expression [negative correlation], and it will be close to 0 if they are unrelated).

Then, based on the thresholds $1 > T_1 > T_2 > 0$, a gene pair $(g_a,g_b)$ is classified into one of the categories shown in Table.

| | $\Phi_{\text{ab2}} \geq T1$ | $\Phi_{\text{ab2}} \leq -T1$ | $\Phi_{\text{ab2}} \in [-T2,T2]$ |
|---|---|---|---|
| $\Phi_{\text{ab1}} \geq T1$ | NC | P2N | $\text{P}_\text{l}$ |
| $\Phi_{\text{ab1}} \leq -T1$ | N2P | NC | $\text{N}_\text{l}$ |
| $\Phi_{\text{ab1}} \in [-T2,T2]$ | $\text{P}_\text{g}$ | $\text{N}_\text{g}$ | NC |

where NC is no change in the relationship, $\text{P}_\text{g}$, $\text{N}_\text{g}$ are gain of positive and negative correlations, respectively, $\text{P}_\text{l}$, $\text{N}_\text{l}$ are loss of positive and negative correlations, respectively, and P2N, N2P are the flip of correlations from positive to negative and negative to positive respectively.

These correlation gain/loss ($\text{P}_\text{g}$, $\text{P}_\text{l}$, $\text{N}_\text{g}$, and $\text{N}_\text{l}$) define a graphical structure of the changed coexpression. The genes with the most changes in correlations with the other genes from $G_1$ to $G_2$ or the most neighbors in changed coexpression graph are selected to be coexpressed differentially.

For example, from Figure 31.14, the gene in the top graph has six neighbors in normal tissues but lost five of them in tumor tissues as shown by the broken lines. In contrast, the gene shown in the bottom graph lost only one out of five neighbors from normal to tumor tissues. This implies that the gene in the top graph is more important compared with the gene in the bottom graph.

The algorithm defines *base genes* as the genes that exhibit the most changes in correlations and the *neighborhood genes* that demonstrate changed correlation with a base gene. The base genes are filtered out further based on the neighborhood reduction strategy. A base gene is eliminated if 70% of its neighborhood is covered by any of the base genes of a better score thus eliminating the redundant base genes and resulting in unique neighborhoods. Each base gene along with its neighborhood now is considered to be one set of a differentially coexpressed gene set.

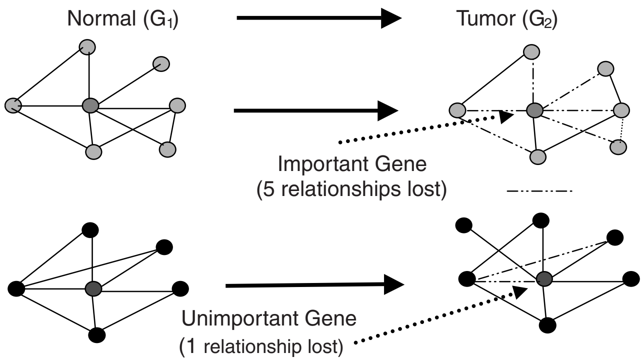


**Figure 31.14** Illustration of the differential friendly neighbors algorithm. It shows two genes and their neighborhoods formed by their FNs in normal and tumor groups. The top one shows a loss of five FNs from normal to tumor groups, whereas the bottom one shows a loss of only one FN. By the DiffFNs algorithm, the gene in the top is more important, and it could be a base gene.

## ACKNOWLEDGMENTS

I thank Lance Miller and Edison T Liu who introduced me to the fascinating field of cancer biology. I thank my colleagues Yew Kok Lee, Max Fun and Huaien Luo for their valuable suggestions on the manuscript. The research was supported by the Genome Institute of Singapore and the Agency for Science Technology and Research (A*STAR).

## REFERENCES

1. R.K.M. Karuturi and V.B. Vega. Friendly neighbours method for unsupervised determination of gene significance in time-course microarray data'. *Proceedings of the International Conference on Bioinformatics and Bio-Engineering (BIBE),* Taichung, Taiwan, 2004.
2. R.K.M. Karuturi, S. Wong, W-K. Sung, and L.D. Miller. Differential friendly neighbours algorithm for differential relationships-based gene selection and classification using microarray data. *Proceedings of the 2006 International Conference on Data Mining (DMIN'06)*, Las Vegas, NV, 2006.
3. D. Kostka and R. Spang. Finding disease specific alterations in the co-expression of genes. *Bioinformatics*, 20:i194–i199, 2004.
4. L.D. Miller, J. Smeds, J. George, V.B. Vega, S. Klaar, P. Hall, Y. Pawitan, A. Ploner, L. Vergara, E.T-B. Liu, *et al.* An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects and patient survival'. *Proc Natl Acad Sci U S A*, 102:13550–13555, 2005.
5. C. Prieto, M.J. Rivas, J.M. Sanchez, J. Lopez-Fidalgo, and J. De Las Rivas. Algorithm to find gene expression profiles of deregulation and identify families of disease-altered genes. *Bioinformatics*, 22(9):1103–1110, 2006.

6. V.G. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionising radiation response. *Proc Natl Acad Sci U S A*, 98(9):5116–5121, 2001.

7. J. Zhang, X. Liu, A. Datta, K.R. Govindarajan, W. Leong Tam, J. Han, J. George, C.W. Wong, K. Ramnarayanan, T.Y. Phua, *et al.* RAB11FIP1/RCP is a novel breast cancer promoting gene with Ras activating function. *J Cline Invest*, 119(8):2171–2183, 2009.

8. H. Li and R. Krishna Murthy Karuturi. Significance analysis and improved discovery of disease specific differentially co-expressed gene sets in microarray data. *Int J Data Min Bioinform*. To appear.

9. S.S. Hoon, Y. Yiting, L.C. Ho, K.R. Krishna M, W. Vanaporn, T. Apichai, C.H. Hoon, C. Ong, P.S. Suppiah, G. Tan, *et al.* The core and accessory genomes of urkholderia pseudomallei: Implications for human melioidosis, *PLoS Pathog*, 4(10):e1000178, 2008.

10. J. Tan, L. Zhuang, X. Jiang, K.R.K. Murthy, and Q. Yu. ASK1 is a direct target of E2F1 and contributes to histone deacetylase inhibitor-induced apoptosis through a positive feedback regulation on E2F1 apoptotic activity. *J Biol Chem*, 281(15):10508–10515, 2006.

11. H. Luo, Y. Sudibyo, and K.R.K. Murthy. *Weighted top Score Pair Method for Gene Selection and Classification, in the LNBI of Pattern Recognition in Bioinformatics (PRIB).* Oxford University Press, Melbourne, Australia.

12. C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, New York, 1995.

13. G.K. Kanji. *100 Statistical Tests*. Sage, Thousand Oaks, CA, 2006.

14. A. Callegaro, D. Basso, and S. Bicciato. A locally adaptive statistical procedure (LAP) to identify differentially expressed chromosomal regions. *Bioinformatics*, 22(21):2658–2666, 2006.

15. J.W. Ho, M. Stefani, C.G. dos Remedios, and M.A. Charleston. Differential variability analysis of gene expression and its application to human diseases. *Bioinformatics*, 24(13):i390–i398, 2008.

16. A.E. Teschendorff, A. Naderi, N.L. Barbosa-Morais, C. Caldas. PACK: Profile analysis using clustering and kurtosis to find molecular classifiers in cancer. *Bioinformatics*, 22(18):2269–2275, 2006.

17. I.K. Kitsas, L.J. Hadjileontiadis, and S.M. Panas. Discrimination of single and multiple human transmembrane proteins using kurtosis and morphological analysis. *Conf Proc IEEE Eng Med Biol Soc*, 2008:1351–1354, 2008.

18. L. Li, A. Chaudhuri, J. Chant, and Z. Tang. PADGE: Analysis of heterogeneous patterns of differential gene expression, *Physiol Genoms*, 32(1):154–159, 2007.

19. A.C. Tan, D.Q. Naiman, L. Xu, R.L. Winslow, and D. Geman. Simple decision rules for classifying human cancers from gene expression profiles. *Bioinformatics*, 21(20):3896–3904, 2005.

20. A. Aggarwal, D.L. Guo, Y. Hoshida, S.T. Yuen, K.M. Chu, S. So, A. Boussioutas, X. Chen, D. Bowtell, H. Aburatani, *et al.* Topological and functional discovery in a gene co-expression meta-network of gastric cancer. *Canc Res*, 66:232–241, 2006.

21. J.R. Pollack, T. Sørlie, C.M. Perou, C.A. Rees, S.S. Jeffrey, P.E. Lonning, R. Tibshirani, D. Botstein, A.L. Børresen-Dale, and P.O. Brown. Microarray analysis reveals a major direct role of DNA copy number alteration in the transcriptional program of human breast tumors. *Proc Natl Acad Sci U S A*, 99(20):12963–12968, 2002.

22. J.L. Morrison, R. Breitling, D.J. Higham, and D.R. Gilbert. GeneRank: Using search engine technology for the analysis of microarray experiments. *BMC Bioinformatics*, 6:233, 2005.
23. A. Altimari, M. Fiorentino, E. Gabusi, E. Gruppioni, B. Corti, A. D'Errico, and W.F. Grigioni. Investigation of ErbB1 and ErbB2 expression for therapeutic targeting in primary liver tumours. *Dig Liver Dis*, 35(5):332–338, 2003.
24. J.G. Hodgson, K. Chin, C. Collins, and J.W. Gray. Genome amplification of chromosome 20 in breast cancer. *Breast Canc Res Treat*, 78:337–345, 2003.
25. G.K. Smyth. Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Stat Appl Genet Mol Biol*, 3(1):3, 2004.
26. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J Roy Stat Soc. S B,* 39(1):1–38, 1977.
27. J.C. Barrett and E.S. Kawasaki. Microarrays: The use of oligonucleotides and cDNA for the analysis of gene expression. *Drug Discov*, 8:134–141, 2003.
28. M. Schena, R.A. Heller, T.P. Theriault, K. Konrad, E. Lachenmeier, and R.W. Davis. Microarrays: biotechnology's discovery platform for functional genomics. *Trends Biotechnol*, 16:301–306, 1998.
29. H. Akaike. A new look at the statistical model identification. *IEEE Trans Automat Contrl*, 19(6):716–723, 1974.
30. G.E. Schwarz. Estimating the dimension of a model. *Ann Stat*, 6(2):461–464, 1978.
31. D.N. Joanes and C.A. Gill. Comparing measures of sample skewness and kurtosis. *J. Roy Stat Soc D*, 47(1):183–189, 1998.
32. C. Torrence and G.P. Compo. A practical guide to wavelet analysis. *Bull Am Meteorol Soc*, 79:61–78, 1998.
33. G. Strang. *Introduction to Linear Algebra*, Section 6.7. 3rd edition. Wellesley-Cambridge Press, Wellesley, MA, 1998.
34. D.G. Albertson, C. Collins, F. McCormick, and J.W. Gray. Chromosome aberrations in solid tumors. *Nat Genet*, 34:369–376, 2003.
35. K. Rennstam, S.M. Ahlstedt, B. Baldetorp, P.O. Bendahl, A. Borg, R. Karhu, M. Tanner, M. Tirkkonen, and J. Isola. Patterns of chromosomal imbalances defines subgroups of breast cancer with distinct clinical features and prognosis. A study of 305 tumors by comparative genomic hybridization. *Canc Res*, 63:8861–8868, 2003.
36. E. Hyman, P. Kauraniemi, S. Hautaniemi, M. Wolf, S. Mousses, E. Rozenblom, M. Ringner, G. Sacter, O. Momi, and A. Elkahloun, *et al.* Impact of DNA amplification on gene expression patterns in breast cancer. *Canc Res*, 62:6240–6245, 2002.
37. W.E. Johnson, C. Li, and A. Rabinovic. Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics*, 8(1):118–127, 2007.
38. E. Herrmann. Local bandwidth choice in kernel regression estimation. *J Graph Comput Stat*, 6:35–54, 1997.
39. Wikipedia: The free encyclopedia. http://en.wikipedia.org/wiki/Main_Page.
40. A.J. Dobson. *An Introduction to Generalized Linear Models*, 2nd edition. Chapman & Hall, New York, 2008.

# VI

# ANALYSIS OF GENOMES

# 32

# COMPARATIVE GENOMICS: ALGORITHMS AND APPLICATIONS

Xiao Yang and Srinivas Aluru

## 32.1 INTRODUCTION

“Comparative genomics” is a constantly evolving term that refers to intra- or interspecies comparisons. It involves but is not restricted to the research areas of ortholog assignment, synteny detection, gene cluster detection, multiple genome alignment, rearrangement analysis, ancestral genome reconstruction, and gene or speciation tree construction. We consider the first three as the “upstream” problems, whereas the rest as the “downstream” problems because the solutions of the former typically are used as inputs to the latter. However, the advance of new algorithmic approaches makes this distinction more ambiguous as they can consider multiple problems concurrently.

The long-term goal of comparative genomics is to understand evolution. From a microlevel, researchers want to understand how nucleotide sequences evolve (*e.g.*, rate of point mutations), and from a macrolevel, how genes, regulatory elements, and networks, or metabolic pathways evolve (*e.g.*, gene insertions, fusions, the functional change of pathway resulting from gene losses, *etc.*) More specifically, thanks to evolution, comparing multiple genomes better reveals functional elements in a genome [36, 18]: the discovery of genes and other functional units, such as transcription factor biniding sites, siRNAs, and so on. Because functional elements tend to be subjected to negative selection, conserved elements among different genomes

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

with properly chosen evolutionary distances are likely to share similar functions. In addition, comparative genomics is used for studying the genome structure, nucleotide composition, and coding strategy among different species, families, or, genus. These goals are tackled by different research areas mentioned earlier, and each of them will be described briefly in the following.

The ortholog assignment amounts to the functional assignment of genes, although the original definition of ortholgs [23] has little to do with gene functionality. The sharing of similar functionalities between two genes in different species provides a strong biological evidence of their orthologous relationship. One of the major biological applications of orthologs assignment is to predict the functions of genes in a newly sequenced genome by comparing them with genes in a well-annotated genome in which a new gene is assigned the same function as its orthologous partner. At the same time, nonorthologous genes, resulting from duplication events, are differentiated. Computationally, assigning orthologs among multiple genomes is required for several downstream analysis. For instance, phylogenetic tree construction uses a single gene family in which the member genes need to form an ortholog group to correctly reflect speciation; most rearrangement studies rely on an input consisting of a one-to-one gene correspondence between two genomes as the result of ortholg assignment.

As a step further, synteny detection tries to find related genomic regions among multiple genomes. Despite the evolutionary changes over time, genomic regions with the same origin could be inferred by sequence conservation or, at a higher level, by a set of orthologous genes with spacial proximities. On the other hand, the conservation on the genomic region level is superior in inferring orthologs to the conservation at the gene level. Hence, ortholog assignment in syntenies has a higher reliability [13]. Similarly, gene cluster detection—identifying a set of closely located genes—as the candidate for a functional unit, shared by multiple microbial genomes, could use the results of ortholog assignment as the input, or the resulting gene clusters could be used as the input to an ortholog assignment problem.

Multiple genome alignment at the nucleotide level amounts to multiple sequence alignment on the whole genome scale in which involved genomes have to be similar enough for sequence alignment to reflect biological meaning. Point mutations (*e.g.*, synonymous change) could be discovered from the alignment. However, among highly diverged or repetitive genomes, sequence alignment is no longer an appropriate approach. In this case, higher level of genome representations (*e.g.*, conserved sequences, exons, genes, or syntenies) [51, 8, 19, 20], are used. But the optimization goal and how to deal with such complications remain to be open problems.

Multiple genome alignment is closely related to rearrangement analysis, which is usually studied along with ancestral genome reconstruction. One of the clear optimization goals in rearrangement analysis is to convert one genome into another using the parsimonious rearrangement operations (typically genomic inversions, translocations, fissions, and fusions). Genomes under rearrangement studies are usually required to comprise of nonrepetitive units (*e.g.*, one gene per family). This could be realized by ortholog assignment or by multiple genome alignment, which results in a multiway matching of basic units across genomes.

The aim of ancestral genome reconstruction (*e.g.*, [7, 10, 40, 11, 27]) is to infer the nucleotide sequence or gene order for the lowest common ancestor of extant genomes in the speciation tree and, while doing so, to recover the order of evolutionary events. Once this goal is fulfilled, it is not hard to see that solutions for most other problems described earlier will be straightforward. Although multiple optimization problems could be defined, many existing methods rely on rearrangement analysis because the ancestral genome is naturally defined as a median genome such that the summation of the pairwise evolutionary distance between the ancestor and each of the extant genomes is minimized, based on either the breakpoint or on the rearrangement distance.

To limit the scope, we will only address the three upstream problems and leave the rest to the other chapters. The outline of this chapter is as follows: in Section 32.2, we provide some general notations, followed by a discussion of orthologs assignment in Section 32.3. We combine coverage of both synteny and gene cluster detection problems in Section 32.4 because both lines of research use very similar approaches. Conclusions follow in the last section.

## 32.2 NOTATIONS

To be consistent, we introduce some general notations, which will be used throughout the chapter.

Let $\mathcal{G} = \{G_1, G_2, \ldots, G_{|\mathcal{G}|}\}$ be a set of genomic sequences, where $|\mathcal{G}|$ denotes the cardinality of $\mathcal{G}$, and each $G_i$ $(1 \le i \le |\mathcal{G}|)$ could be a chromosomal segment, a chromosome, or a genome. Unless specified otherwise, each genomic sequence $G_i$ will be delineated by a sequence of genes (or genetic markers): $G_i = (g_{i1}, g_{i2}, \ldots, g_{in_i})$, where $n_i$ is the number of genes in $G_i$. We use a lower case letter with superscript or subscript to denote a gene if its genomic location does not need to be specified (*e.g.*, $g$ and $g'$).

Given two genes $g$ and $g'$, let $s(g, g')$ denote their sequence similarity score, which is defined by a user-specified function, and let $t_s$ be the threshold. For any two genes $g_{ij}$, $g_{ik}$ $(1 \le j, k \le n_i)$ on the same genomic segment $G_i$, let $d_g(g_{ij}, g_{ik}) = |k - j| - 1$ denote their relative gene distance (*i.e.*, the number of genes between $g_{ij}$ and $g_{ik}$), and let $d_p(g_{ij}, g_{ik})$ be their physical distance in bases.

Let $\mathcal{U}$ be the universal gene set. Any two genes $g, g' \in \mathcal{U}$ belong to the same gene family if $s(g, g') \ge t_s$. Let $\Sigma = \{1, 2, \ldots, |\Sigma|\}$ be the set of all gene family labels in $\mathcal{U}$, and define function $f : \mathcal{U} \to \Sigma$ to map genes to their respective gene family labels.

## 32.3 ORTHOLOG ASSIGNMENT

We first introduce several terms regarding the evolutionary relationships between two genes [38, 55]. Two genes are *homologs* if their sequence similarity is above a given threshold. Homologs are divided into two types: *orthologs*—genes related

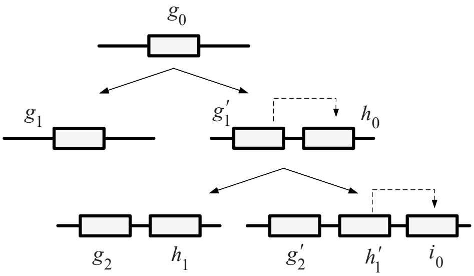


**Figure 32.1** Examples of genes with different evolutionary relationships: homologs, orthologs (main orthologs), and paralogs (in-paralogs and out-paralogs). A solid line denotes a genomic segment in which a gene on the segment is shown as a rectangle. Solid arrows represent speciations, and dotted arrows denote duplications.

through speciation with similar functions[1]—and *paralogs*—genes related through duplication, which may or may not have the same function. Paralogs are classified further into two subtypes: *in-paralogs*—paralogs that developed after speciation—and *out-paralogs*—paralogs that developed before speciation.

An example is given in Figure 32.1 showing these different relationships; $g_0$ is an ancestral gene, which gives rise to an orthologous gene pair, $g_1$ and $g_1'$, after speciation. The gene $h_0$, duplicated from $g_1'$, is also orthologous to $g_1$ by definition. To differentiate the order of inception, $g_1$ and $g_1'$ are termed *main orthologs*. Genes $g_1'$ and $h_0$ are in-paralogs with regard to $g_1$. The second speciation results in $g_2$ and $h_1$ on one genomic segment and in $g_2'$ and $h_1'$ on the other. Because the speciation event comes after duplication, $g_2$ and $h_1$ are out-paralogs with respect to $g_2'$ and $h_1'$, and vice versa. Likewise, $h_1$ is orthologous to both $h_1'$ and $i_0$ but forms main orthologs with the former. All genes are homologs in the figure.

A major goal of ortholog assignment among multiple genomes is to characterize proteins with unknown functions. For instance, to annotate a newly sequenced genome, a gene could be assigned the same function as its orthologous genes in a known species. Among other applications, orthologs are used in many phylogenetic algorithms to infer gene or speciation trees; and to make the rearrangement analysis tractable, ortholog assignment between two genomes is typically required. However, ortholog assignment in multiple genomes is a challenge because of several complications: point mutations make orthologs dissimilar on a sequence level, and frequent gene losses and duplications make it impossible to differentiate paralogs and orthologs using sequence similarity, as well as other forces such as horizontal gene transfer, gene fusions, and so on.

Numerous algorithms (*e.g.*, [63, 55, 62, 39, 13, 17, 67, 16, 24, 25]) have been developed to identify orthologous genes among multiple genomes. We classify them into three types, which are based on sequence similarity, phylogeny, and

[1] Adapted from the original definition by Fitch [23], which has no functional similarity requirement. Orthologs could not be verified biologically without this requirement.

rearrangement. Note that the boundaries of these classifications are not absolute, and some methods take into consideration multiple criteria.

Sequence similarity quantification is usually adopted to establish rough homologous relationships between all pairs of genes in which the method used need not be very precise but relatively fast. For instance, BLAST is a popular choice. More accurate and time-consuming methods could be used such as pairwise sequence alignment. Ortholog clustering methods based on phylogeny rely on more delicate sequence similarity functions between genes (*e.g.*, multiple sequence alignment) followed by a tree construction method (*e.g.*, parsimony or maximum likelihood), using alignment results. This tends to be more time consuming, especially on gene families with a large number of genes. Methods based solely on sequence similarity or phylogeny share the same assumption that local mutations of genes correctly reflects evolution so that orthologous genes tend to be more similar to each other than to other genes on different genomes. When this assumption does not hold (*e.g.*, frequent gene duplications and losses in plant genomes), ortholog gene assignment based on positional information on genomes as well as adjacent gene content is considered more reliable (*e.g.*, rearrangement-based approach).

### 32.3.1 Sequence Similarity-Based Method

A sequence similarity-based method could be generalized to a gene based approach graph in which each vertex in the graph denotes a gene, and genes connected by an edge have a high similarity score ($\geq t_s$). Typically, the similarities between two genes are determined empirically based on BLAST outputs using two criteria: e-values of high scoring pairs (HSPs) and the ratio between alignment length and the length of the compared sequences. Then, a clustering algorithm is applied to the gene graph in which the resulting clusters denote orthologs groups.

Given two genomes $G_i$ and $G_j$, $g_{ik}$ and $g_{jl}$ are termed *reciprocal best hits* (RBHs) if

$$s(g_{ik}, g_{jl}) = \max_{1\leq l'\leq n_j} (s(g_{ik}, g_{jl'}))$$
$$s(g_{jl}, g_{ik}) = \max_{1\leq k'\leq n_i} (s(g_{jl}, g_{ik'}))$$

where the scoring function is given by BLASTing the second gene against the first one in the function. Note that the need to use BLAST twice for comparing two genes is a result of the asymmetric score function used by BLAST. RBHs are typically used to infer orthologs between two genomes.

We describe several popular algorithms using the above ideas. The Clusters of Orthologs Groups (COG) [63] is one of the first widely used ortholog classification databases that compare genes in bacteria, archaea, and eukaryotes. After applying all-vs-all BLAST comparisons to establish the pairwise gene similarities between any two genomes, a gene graph could be created with each edge representing RBHs. Then, the clustering step is to identify cliques with a minimum size of three under the assumption that clusters with this size are likely preserving similar functions in different species. Nevertheless, it is necessary to identify orthologs between just two genomes, which could not be accommodated by COG. To achieve this goal, INPARANOID [55] was developed, whose results serve as a benchmark that most newly developed algorithms are compared with.

The major contribution of INPARANOID is to classify in-paralogs in each genome, which are co-orthologs for genes with the same function in another genome. Given two genomes $G_i$ and $G_j$, the gene graph consists of two types of edges: (i) all-vs-all BLAST self-comparisons ($G_i$ vs. $G_i$ and $G_j$ vs. $G_j$) result in the edges connecting genes with a high similarity score within each genome; (ii) all-vs-all BLAST comparisons of genes between $G_i$ and $G_j$ result in edges connecting genes with RBHs. Genes connected by RBH edges are candidates of main orthologs. Under the assumption that in-paralogs are more similar to each other than to their orthologous counterparts, genes $g$ and $g'$ connected by a type (i) edge are classified as in-paralogs if their similarity score is higher than a type (ii) edge connecting $g$ or $g'$, if any.

A more sophisticated clustering algorithm is described in [67]. The initial input of multiple genomes $\mathcal{G}$ is modeled as a multipartite graph $G = (V, E)$, where vertex set $V$ denotes genes, and edges only connect genes $g_{ij}$ and $g_{lk}$ if $i \neq l$ and $s(g_{ij}, g_{lk}) \geq t_s$. An ortholog clustering step on this graph follows, which takes into consideration two criteria: (i) similarities between genes within and outside the target cluster, and (ii) phylogenetic relationships between genomes. These are measured quantitively in the following linkage scoring function:

$$\pi(g_{ij}, C) = \sum_{l=1, l \neq i}^{|\mathcal{G}|} d_{\text{phy}}(G_i, G_l)\Big( \sum_{k=1, g_{lk} \in C}^{n_l} s(g_{ij}, g_{lk}) - \sum_{k=1, g_{lk} \notin C}^{n_l} s(g_{ij}, g_{lk})\Big) \quad (32.1)$$

where $d_{phy}(G_i, G_l)$ denote the phylogenetic distance between genomes $G_i$ and $G_l$, and $C$ is the targeting ortholog cluster. The first and the second summations capture the within gene cluster similarities and the outside gene cluster similarities, respectively. The optimization problem is to identify a cluster $C^*$, which maximizes function $F(C) = \min_{g \in C} \pi(g, C)$ for any $g \in C$, with $C$ being a subset of all genes.

It can be proven that using function $F$, there exists a unique gene cluster $C^*$ such that $C^*$ maximizes function $F$, and $F(C^*) \geq F(C')$ for any $C' \supset C^*$. Based on this observation, to identify $C^*$, we could start with all genes as the initial cluster then iteratively remove genes contributing the lowest scores (defined by Equation 32.1) to the cluster until all genes are removed or the score of the newly generated cluster equals to zero. Let $H_i$ be the cluster generated in the $i$th iteration and $\mathcal{H}$ be the set of all clusters generated by the above algorithm ($A$), then:

$$C^* = \underset{H_i \in \mathcal{H}}{\text{argmax}}\, F(H_i) \quad (32.2)$$

Naturally, the following algorithm ($B$) identifies all ortholog gene clusters in the input until all genes are removed (or stop until no cluster has size $\geq 2$).

1. Apply algorithm $A$ to the input $\mathcal{G}$.
2. Update $\mathcal{G}$ by removing all genes from $C^*$.

Algorithm $B$ partitions the input data into a set of orthologous gene clusters $\{C_1, C_2, \ldots, C_m\}$, satisfying $F(C_1) > F(C_2) > \ldots F(C_m)$. With an efficient data structure, the overall running time of $O(|E| + |V|)$ is achieved.

### 32.3.2 Phylogeny-Based Method

If the mutation rates of gene sequences in different species correctly reflect the timing of speciation, then the phylogeny-based method could be the most intuitive approach for inferring gene orthology because the gene tree should be consistent with the speciation tree. However, this assumption may not hold true because of gene duplications and losses [13], and the quality of phylogenetic tree inference worsens along with the divergence of genomic sequences regardless of this assumption.

The solution of the following problems control the quality of the phylogeny-based method:

- Gene family construction: to identify the candidate genes that will be used for phylogenetic tree construction. It is necessary to use a stringent threshold for determing gene similarity because genes belonging to different gene families could share the same domains. This step typically generates a pairwise distance matrix of genes.
- Multiple sequence alignment: to infer point mutations that different genes within the same family went through. The solution of this nondeterministic polynomial (NP)-complete problem provides an overall picture of gene sequence evolution. To improve the quality of the final tree, many refined alignment methods are used such as T-COFFEE [42] and MUSCLE [22], which take consideration protein secondary structure.
- Phylogenetic tree construction: to establish evolutionary relationships among genes. Two categories of methods are used, distance based (*e.g.*, neighbor-joining tree) and character-based (*e.g.*, maximum likelihood tree). The former only needs a pairwise distance matrix regardless of how it is derived. However, the latter requires the results from multiple sequence alignment; hence, it is more time-consuming.
- Tree evaluation: to assess the significance of the tree compared with a random tree, typically using a bootstrap technique. For instance, Storm *et al.* [62] used a sampling with replacement strategy on the columns of a multiple sequence alignment to generate new random alignments with the same number of columns. By applying the same tree construction methods on these random alignments, a support value is derived for the original tree.

Though refined multiple sequence alignments and character-based methods are considered to yield better results in practice, they are time consuming and are limited to a small number of sequences. Recent development of parallelization strategies [46, 74] is underway to address these problems.

### 32.3.3 Rearrangement-Based Method

There are multiple functions to calculate the distance between two genomes (*e.g.*, the *breakpoint distance*—the number of times in which two genes are adjacent on one genome but not on the other—and the *reversal distance*—the minimum number of genomic inversions needed to transform one genome to the other). Efficient algorithms have been developed to tackle these problems (*e.g.*, [1, 4, 10, 31, 35, 58, 68]). Although these methods are typically used for studying genome rearrangement and ancestral genome reconstruction, they have been useful in studying ortholog assignment in two closely related genomes.

However, evolutionary complications such as gene duplications and losses prevent the direct applicability of these methods to resolve the ortholog assignment problem because two genomes under rearrangement studies are required to have an equal number of genes with a one-to-one correspondence. Therefore, the rearrangement analysis methodology needs be modified for ortholog assignment.

Exemplar gene identification [56] between two genomes is one of the first direct applications of rearrangement analysis. This also could be considered as the main ortholog assignment problem. The optimization goal is to identify an ortholog assignment that minimizes a chosen distance between two genomes (*e.g.*, breakpoint distance). In this particular problem, an exact branch-and-bound algorithm is used to enumerate all possible one-to-one gene matching between two genomes. During enumeration, the minimum distance maintained as the bound to filter unnecessary exploration of some branch of the spanning tree helps expedite the process.

A series of algorithms [16, 24, 25] have been developed to identify main orthologs and in-paralogs using sequence similarity as well as the genomic rearrangement information in pairwise or among multiple genomes.

We briefly discuss the ideas behind the basic algorithm, termed SOAR [24], then present its workflow. Bacause the main rationale of the extended algorithms, MSOAR [24] and MultiMSOAR [25], remains the same, we will describe their improvements over the basic algorithm.

SOAR is based on the idea of sorting signed permutations by reversals [1, 31, 35]. It defines a variant of this optimization problem, termed signed reversal distance with duplicates (SRDD), to accommodate gene duplications. Given two genomes $G_1$ and $G_2$, if a matching exists between genes in $G_1$ and $G_2$, then any efficient algorithm listed above could be used to calculate their reversal distance. The optimization problem SRDD is used to identify such a matching between $G_1$ and $G_2$, which are permutations with redundant numbers (*i.e.*, duplicated genes) such that the reversal distance between $G_1$ and $G_2$ based on this matching is the minimum among all possible matchings. Supposedly, this matching gives an ortholog assignment between $G_1$ and $G_2$.

SRDD is proven to be NP-complete. Therefore, SOAR approximates the solution of SRDD and reduces the complexity of the problem in three stages, in which the approximation of SRDD in each stage is within a bounded distance to the optimum solution.

- Stage 1. Local ortholog assignment. For example, if $g_{1i}$ is a duplicated gene, whereas $g_{1,i-1}$ and $g_{1,i+1}$ are unique, and there exits a matching $g_{2,j-1}, g_{2,j}, g_{2,j+1}$ in $G_2$ corresponding to $g_{1,i-1}, g_{1,i}, g_{1,i+1}$, then this matching gives a local ortholog assignment between the corresponding three gene pairs.
- Stage 2. Coarse-grained partitioning. If a set of genes occur in both $G_1$ and $G_2$ in the same order and orientation, then two genomic regions spanned by this gene set is likely orthologous. Because the larger the gene set, the more confidence we will have to infer their orthologous relationship, the goal is to find a minimum number of such gene sets that partitions both $G_1$ and $G_2$.
- Stage 3. Fine-grained matching. Stage 2 does not guarantee a unique assignment between duplicated genes in $G_1$ and $G_2$, and hence, further analysis is needed. Borrowing from the idea of the approach to convert one genome to another by cycle decomposition in a breakpoint graph [31], the result from the former step is converted to a breakpoint graph, and the goal is to maximize the number of cycles in the graph by removing intergenome edges.

Because both problems in stages 2 and 3 are NP-complete, heuristic algorithms are used. After fulfilling these steps, $G_1$ could be transformed to $G_2$ with reversal operations, which defines a one-to-one gene correspondence (*i.e.*, orthologs assignment).

The main idea remains the same in MSOAR (*i.e.*, to find a sequence of parsimonious operations to transform one genome into another), in which the transformation itself defines the ortholog assignment. Two types of operations are considered: duplications and rearrangement events, which could be approximated by reversals.

Algorithmically, several changes and additions of SOAR have been accommodated in MSOAR to consider in-paralogs and to handle multichromosomal genomes:

1. Two new suboptimal rules have been added in stage 1 to improve the run time.
2. Derive a maximum gene matching between two genomes, used as the input of stage 2, and ignore the rest of the genes.
3. Multiple chromosomes of a genome are concatenated in random order, to be converted to a pairwise comparison problem. Convert the two genomes to a breakpoint graph, remove intergenome edges to maximize the number of cycles, and minimize the number of paths with two endpoints in the same genome.
4. In-paralogs are identified as the duplicated genes that are not involved in the matching.

MSOAR gives a solution to a main orthologs assignment between a pair of genomes, which could be invoked $\binom{n}{2}$ times to derive the pairwise main orthologs across $n$ genomes. The assignment of main orthologs in more than two genomes, referred to as main orthologs clusters, is addressed in MULTIMSOAR.

Initial ortholog clusters are created by single linkage clustering using pairwise ortholog assignment. Initial clusters could be divided into two types: nonambiguity and ambiguity clusters. A nonambiguity cluster—each genome contributes at most one gene—is defined as a main ortholog cluster. In an ambiguity cluster—two or more genes in the cluster belong to the same genome—MultiMSOAR attempts to identify a multiway matching as the solution of ortholog assignment, and duplicated genes not in the matching are regarded as in-paralogs. To initiate the matching problem, dummy genes are introduced in each of the ambiguity cluster so that the number of genes from each genome involved in the cluster are the same. Because even the three-matching problem is NP-complete, a heuristic multiway matching solver is used.

## 32.4 GENE CLUSTER AND SYNTENY DETECTION

The term *synteny* has gained new meanings from genetics study to comparative genomics study [50]. In genetics, a *synteny* refers to multiple genes on the same chromosome whether or not they are genetically linked. In comparative genomics, however, no rigorous definition exists of what forms a synteny, neither biologically nor computationally. We roughly outline the existing usages of "synteny" in comparative genomics in the following two contexts:

- *C1*: large-scale genomic alignment [20, 44, 51]. Because sequence alignment dealing with whole genomes are difficult to handle, identification of syntenies—smaller genomic regions with a high sequence similarity—help reduce the complexity of the overall problem, and the resulting syntenies serve as the input to the follow-up analysis such as whole genome alignment.
- *C2*: derive evolutionarily related genomic regions in multiple genomes using genetic markers [65, 12, 29, 28, 30, 71, 59]. A genetic marker could be a gene or a piece of DNA sequence or even a single nucleotide polymorphism (SNP) with a known location on a chromosome, which is associated with some specific trait. A synteny refers to a set of genomic regions sharing similar marker set in proximate locations.

$C1$ is limited to closely related genomes so that on the primary sequence level, they are similar enough such that meaningful alignments could be generated; meanwhile, enough divergence is needed to differentiate conserved components from "junk DNA segments"[2]. However, local genomic similarity does not directly reveal their evolutionary relationships if they are from repetitive regions. Moreover, it is typically the case that unique conserved region shared by two genomes are usually functional components (*e.g.*, exons), that could be used as genetic markers; therefore,

[2] Diverged genomes could be compared with different metrics (*e.g.*, [15]).

we interleave the meaning of synteny from $C1$ and give the following definition to reflect evolution:

**Definitio 32.1** *A synteny is a set of extant genomic regions, which evolved from the same genomic region of an ancestral genome after speciation or large-scale duplication.*

This definition provides a template in which an embodiment for a specific study will make the definition precise by addressing the following factors:

1. Operand: the entity in describing a genomic region
2. Operator: evolutionary modifications that impose on a genomic region; this specifies the term "evolve" in the definition
3. Result: the status of the ancestral genomic region at some time point of evolution. This defines the ancestor–descendant relationship, which could be defined by a distance function

A genomic region could be delineated by different entities as long as the underlying genomic sequence could be identified unambiguously. In practice, several ways exist to this:

- Use the primary sequence, (*i.e.*, nucleotide composition), where the boundary is accurate to the base level, given as the start and end position on the chromosome.
- Use higher level of annotations (*e.g.*, genes), where the boundary is specified by the first and the last annotated genes in the region, or use markers for genetic linkage studies, where the boundary could not be specified accurately except through their relative order (defined by genetic distance) within a region.

To give an example, let the operands be genes. Then, an ancestral genomic region could be represented as: $A = (a_0, a_1, \ldots, a_{n_A})$ with $n_A$ number of genes. In this case, the operators could be chosen as the evolutionary events, such as gene insertions, duplications, losses, genomic inversions, and so on. Finally, we need to specify what could be a resulting genomic region $G$ that is considered the descendant of $A$. To fulfill this purpose, a distance fuction could be defined between $A$ and $G$ by restricting the number of times different operators are applied to $A$. For example, let $A = (a_0, a_1, \ldots, a_9)$ be an ancestor, transformed to $(a_0, a_1, a_2, a_3, a_4)$ after one segmental gene loss, to $(a_0, b_0, a_1, a_2, a_3, b_1, a_4)$ after two gene insertions, to $(a_0, b_0, a_2, a_3, b_1)$ after two local gene losses, and finally to $(a_0, a_2, b_0, a_3, b_1)$ after one genomic inversion. Therefore, $G$ is considered a descendant from $A$ if we allow six evolutionary events.[3]

This description is under the assumption of knowing $A$ and the path of evolution (*e.g.*, incremental changes applied to $A$), which is difficult to recover if it is even possible. We believe that Definition 32.1 generalizes prior methods for synteny detection in line with $C2$ and clearly reflects evolution, and we regard it as the biological ideal to seek.

Although methods for synteny detection normally applies to animal or plant genomes, *gene cluster detection* starts the application in microbial genomes, with the assumption that a set of closely located genes on different bacterial chromosomes are likely members in a functional unit or pathway [43, 45, 47, 61]. Unlike synteny detection in which different research groups have their own definition, several well-defined models exists for gene cluster detection (*e.g.*, common interval model and gene-teams model).

Although it seems that synteny detection and gene cluster detection apply to different types of organisms and with different biological goals, both lines of research share a significant algorithmic goal while modeling these two problems. And it is our intention to put both lines of research in the same section so that the readers could see this from an algorithmic point of view, and we believe Definition 32.1 could be generalized to both.

In the following, we will describe models and algorithms used in synteny detection and gene cluster detection, respectively.

### 32.4.1 Synteny Detection

As discussed earlier, because the path of evolution is unknown, other criteria are needed to determine whether two genomic regions $G_1$ and $G_2$ belong to the same synteny. The assumption is that if we could identify two sequences of a certain number of markers with similar order and spacing in $G_1$ and $G_2$, which is unlikely to occur by chance, then, $G_1$ and $G_2$ are related evolutionarily (*i.e.*, through speciation or large scale duplication), hence forming a synteny.

Generally speaking, two main criteria are considered in the literature: order of genetic markers and their proximity. When genetic markers are genes, additional criteria such as strand location or direction of transcription are used.

Conservatively, two genomic regions $G_1$ and $G_2$ are inferred to be in a synteny if they share a collinear sequence of markers; marker sequence $M_1 = (m_{11}, m_{12}, \ldots, m_{1n})$ in $G_1$ is *collinear* to $M_2 = (m_{21}, m_{22}, \ldots, m_{2n})$ in $G_2$, if $m_{1i}, m_{2i}$ $(1 \leq i \leq n)$ are the same type of markers. Although collinearity provides a strong evidence of the same origin of evolution, the order frequently is disrupted by a rearrangement events [51, 72]. Hence, whether to adopt a more stringent order constraint so that we have a higher confidence in the detected syntenies to be related evolutionarily or to account for rearrangement events so that more distal

[3] For simplicity, evolutionary events are not defined precisely because their meanings are straightforward in the context.

related regions could be revealed at risk of discovering spurious syntenies is a tradeoff in synteny detection methods. The choice of proximity criterion is arbitrary in which the typical choices are the physical distance or marker distance between two markers of interest (*e.g.*, $d_p$ and $d_g$ for the gene markers).

We term a synteny consisting of two genomic regions a *pairwise synteny* and that consisting of more than two genomic regions a *multisynteny*. Undoubtedly, a multisynteny encodes more information on evolutionary relationships than what could be derived from its constituent pairwise syntenies. For instance, a multisynteny consisting of three genomic regions $G_1, G_2, G_3$, where $G_1, G_2$ and $G_2, G_3$ form two pairwise syntenies, whereas $G_1$ and $G_3$ share no homologous gene pairs, reveals a differential gene loss [60]. A multisynteny usually is constructed from a set of pairwise synteny using transitive relationships (*e.g.*, if $G_1$ is syntenic to $G_2$ and $G_2$ is syntenic to $G_3$, then $G_1$ is syntenic to $G_3$), whether or not $G_1$ shares some markers with $G_3$. Weaker syntenies could be revealed using this incremental approach. For instance, more ancient large-scale duplications are revealed by comparing the rice genome with the *Arabidopsis* genome than the self-comparison of 12 chromosomes of the rice genome [66].

It is noteworthy to mention that there are two implicit assumptions for identifying a multisynteny: (i) A subset of species under comparison are closely related, whereas some more distal species exist that serve as the out-group in the evolutionary tree so that with a set of carefully chosen out-groups, we could validate the hypothesis of more ancient evolutionary events such as whole genome duplication. (ii) The set of genomic regions forming a multisynteny are of similar size (*e.g.*, the number of markers or the length of the genomic sequence).

Synteny detection methods typically use a combination of the following techniques depending on how they handle the marker order and proximity constraints: statistical methods [29, 30], the dot-plot matrix-based approach [65, 12], and the dynamic programming approach [28, 59].

Because it is difficult to explicitly define the "proximity" constraint, a natural approach for detecting syntenies between two genomic sequences is to evaluate how rarely a sequence of markers could be observed, as addressed by statistical analysis. LINEUP [29] was applied to maize genetic maps with the following three constraints: (i) the minimum number of markers involved, (ii) the order of markers in which the disruption of collinearity is allowed, and (iii) the maximum distance between two markers of interest, specified by the number of marker insertions.

A basic algorithm is given with the input consisting of two genetic marker sequences for chromosomes $G_1$ and $G_2$:

1. Remove unique markers from $G_1$ and $G_2$.
2. Using $G_1$ as the reference, find collinear subsequences between $G_1$ and $G_2$, allowing no insertions in the reference.
3. For each subsequence in $G_1$, compute a distance score for any candidate subsequence (satisfying the constraints mentioned earlier) in $G_2$, penalizing the insertions and deletions in $G_2$ and choose the subsequence with the highest score.

With $N$ number of input markers, because the number of subsequences in $G_1$ is bounded by $O(N^2)$ and the size of any matching subsequence in $G_2$ is bounded by a constant, this basic algorithm runs in $O(N^2)$ time.

A statistical evaluation with Monte Carlo simulation is applied by repeating the following procedures 1000 times: randomly permute $G_2$, and apply the basic algorithm. The resulting pairwise syntenies from the simulations are binned by their size. Each synteny from the original output is evaluated against a bin with the same size in the simulation. Any synteny contributes to the final result if its score is among the top 5%.

To relax the collinearity constraint, any marker in $C_1$ is allowed to swap with other markers within a limited genetic distance; then each such permutation is compared with $C_2$ in a similar fashion as described in the basic algorithm.

Similar to LINEUP algorithmically, however, CLOSEUP [30] used only the gene proximity and density (number of homologous gene pairs shared within a unit genomic region) constraints to find the initial candidate syntenies followed by a statistical significance test. Because the constraints are less restrictive when compared with LINEUP, CLOSEUP ran into the problem of inability to evaluate syntenies with a larger size because of the limitation of using a Monte Carlo simulation; random shuffling of genes in a chromosome result in a no long conserved region. Therefore, an upper bound (= 10) is set to limit the size of a synteny.

The dot-plot is used widely for visualizing the similarity between two nucleotide sequences $S_1$ and $S_2$ by drawing them on the $x$ and $y$ axis in a cartesian coordinate system. A dot is plotted at position $(i, j)$ if and only if $S_1[i] = S_2[j]$, where $S_1[i]$ denotes the nucleotide on the $i$th position of $S_1$. ADHORE [65] and DIAGHUNTER [12] adopted this approach by changing $S_1$ and $S_2$ to the gene sequences $G_1$ and $G_2$ on the $x$- and $y$-axis, and a dot is plotted at $(i, j)$ if the $i$th gene in $G_1$ is homologous to the $j$th gene on $G_2$. The collinearity requirement between $G_1$ and $G_2$ is fulfilled by identifying diagonal or antidiagonal lines on the dot-plot in which a distance threshold is specified for adjacent dots. ADHORE used linear regression to test the fitness of a set of dots to the diagonal or antidiagonal lines followed by a random permutation test for their statistical significance. Any resulting set of collinear genes on the dot-plot form a synteny.

A dot-plot between $G_1$ and $G_2$ also could be modeled as a directed acyclic graph (DAG) $G = (V, E)$, where $v_{ij} \in V$ denotes the dot in position $(i, j)$, and $\overrightarrow{e} = (v_{ij}, v_{kl}) \in E$ exists if $k > i$. Any scoring function reflecting the proximity and the order of the dots could be used, and the goal is to identify any maximal scoring path $p = (v_{i_1,j_1}, v_{i_2,j_2}, \ldots, v_{i_n,j_n})$ such that $i_1 < i_2 < \cdots < i_n$ and $j_1 < j_2 < \cdots < j_n$ or $j_1 > j_2 > \cdots > j_n$. Path $p$ corresponds to the diagonal or antidiagonal lines in the dot-plot. Note that a subpath of the maximal scoring path is also locally maximal; therefore, the solution of the overall problem is depend on the solution of the subproblems, and thus, dynamic programming comes into play (*e.g.*, [28]).

To unveil syntenies consisting of ancient genomic regions as resulting from large-scale genomic duplications followed by frequent gene losses, it is necessary to compare multiple genomes in hopes of identifying some of its constituent well-conserved pairwise syntenies. Then, using the pairwise synteny as the seed, a new round of comparison could be applied, and this process iterates until no more genomic regions could be found. This idea is implemented in a genome-profile-based approach (I-ADHORE) [60, 59], which has a higher sensitivity compared with other methods. In the following, we describe this algorithm for identifying a multisynteny on an input of multiple genomes:

1. ADHORE (described earlier) could be used to identify a significant pairwise synteny.
2. Two genomic regions in the pairwise synteny are converted to two strings $S_1$ and $S_2$ of gene family labels. Alignment is performed on $S_1$ and $S_2$ using dynamic programming (*e.g.*, Needleman–Wunsch algorithm [41]). An alignment profile is constructed from the alignment; if the gene family label $f(g)$ aligns with an indel or with $f(g')$ on the $i$th position, then emit $f(g)$ in the profile at the same position.
3. The profile is treated as a new genomic region; iterate it through the previous steps until no significant pairwise synteny can be found.

Note that, in step 2, if any of the genomic regions is derived from a profile, then multiple alignment is performed instead of pairwise alignment, and the profile generation step is similar.

### 32.4.2 Gene Cluster Detection

Gene cluster detection has been studied intensively during the past decade (*e.g.*, [2, 3, 5, 6, 9, 14, 21, 26, 32, 33, 34, 37, 48, 49, 52, 53, 57, 64, 69]). A variety of formal models have been developed. Because excellent reviews and descriptions of these models already exist [33, 5], instead of repeating them here, we will organize this section as follows: to be self-contained, we will briefly describe some of these models and what have been captured, then we will describe recent progress and development, and finally, we will address several issues that we believe are important and need be considered for future development.

Different from synteny detection, in gene cluster models, gene directionality and strand information generally are overlooked, and instead, the focus is on the gene orders, proximities, and multiplicities. There are several formal models describing what forms a gene cluster, such as the $r$-window model, common interval model, gene teams model (also known as the max-gap cluster model), and median gene cluster model. In this order, each model generalizes the previous one, and intuitively, we are expecting the improvement of the sensitivity. However, this is just one side of the story; better sensitivity does not necessarily mean a better model.

We will start with the gene teams model [6] because of the multiple favorable properties it possesses [33], and so far, the biological results achieved by this model are better or on par with the others.

We regard a *gene cluster* as a set of gene family labels, which is an abstract concept, and an *instance* of this gene cluster is a gene sequence present at some location of a chromosome. The validity of a gene cluster is derived from and supported by its instances. Although the term "gene cluster" has been used for both the abstract meaning as well as for the instances, the ambiguity is manageable. However, we prefer to keep them separate.

***32.4.2.1 Gene Teams Model.*** Given two chromosomes $G_1$ and $G_2$, where $G_i = (g_{i1}, g_{i2}, \ldots, g_{in_i})$ $(1 \le i \le 2)$, a set of gene families $C$ form a $\delta$-gene team (or max-gap cluster) with two instances $I_1 = (g_{1j_1}, g_{1j_2}, \ldots, g_{1j_n})$ and $I_2 = (g_{2k_1}, g_{2k_2}, \ldots, g_{2k_n})$ if the following conditions hold, where $\delta$ is the parameter specifying the maximum allowable distance between two adjacent genes in any instance of the gene team:

1. $\bigcup_{j_1 \le j' \le j_n} f(g_{1j'}) = \bigcup_{k_1 \le k' \le k_n} f(g_{2k'}) = C$
2. $d_g(g_{1j_l}, g_{1j_{l+1}}) \le \delta$ and $d_g(g_{1k_l}, g_{1k_{l+1}}) \le \delta$ for $1 \le l \le n-1$

Bergeron *et al.* [6] developed an efficient algorithm to identify all gene teams in $O(n \log^2(n))$ time between two genomes with $n$ genes each. This is a divide-and-conquer approach consisting of the following ideas: first, transform $G_1$ and $G_2$ into $G_1'$ and $G_2'$ by removing genes that are unique to either of them. Then, split $G_1'$ and $G_2'$ into multiple segments at places where two adjacent genes are more than $\delta$ distance apart because any two such genes will not be in the same team according to the definition. Let the resulting segments be denoted by $\{G_{11}, G_{12}, \ldots, G_{1n_1}\}$ and $\{G_{21}, G_{22}, \ldots, G_{2n_2}\}$ for $G_1'$ and $G_2'$, respectively. Without loss of generality, consider $G_{11}$, if we find that some $G_{2j}$ $(1 \le j \le n_2)$ share the same gene families with $G_{11}$; then they form a team. Otherwise, $G_{11}$ will be split into multiple segments again because some genes exist that belong to different segments of $G_2'$. With the same rationale, each segment generated in some stage is tested against its counterparts, and either the split will take place, or the gene team is found. This algorithm is generalized to $m(> 2)$ genomes with the complexity of $O(mn \log^2 n)$.

This basic algorithm is used in many more realistic biological applications, such as considering gene duplications [32]. Kim *et al.* [37] proposed a more flexible gene teams model, which relaxes the required spatial proximity. They proposed a breadth-first search algorithm to find gene clusters in multiple microbial genomes based on the fact that a gene cluster may exist only among a subset of genomes. Starting from a reference genome as the root of the tree, as more genomes are incorporated, the genomic segment is split into smaller parts based on the common genes, which are represented as the branches of the tree. On the other hand, instead of relaxing the constraints, Zhu *et al.* [73] used a more stringent proximity criterion than the gene teams model, and got comparable results as the gene teams model. This raised the question of whether the gene teams model is too permissive in practice than is necessary. This consideration is consistent with the conclusion drawn by the experiment carried out in [33] with bacterial genomes.

The gene teams model takes into consideration genomic inversions, gene insertions, and duplications. When gene insertions are not allowed (*i.e.*, $\delta = 0$), this model reduces to the common interval model.

Recently, median cluster model [54, 9] was introduced that has a different optimization criteria for gene clusters; given a distance threshold $\delta$, a gene family set $C$ is a median gene cluster if

$$\sum_{i=1}^{k} |C \setminus F(I_i)| + |F(I_i) \setminus C| \leq \delta$$

where $F(I_i)$ denotes the gene family set of instance $I_i$.

ILP [54] is one of the median gene cluster models using an integer linear programming approach. Variations in gene cluster detection, such as the cardinality of a gene cluster, or the presence or absence of a gene cluster in a particular genome, could be modeled with different constraints. Therefore, because of its generality, the common interval model and the gene teams model could be captured by varied ILP formalization. Although ILP is a known NP-complete problem, it has been studied intensively in computer science community and efficient approximation solvers exist. B*ö*cker *et al.* [9] proposed several algorithms to identify median gene clusters by identifying a filter, which is an instance of a gene cluster. Using this filter, all other instances in the dataset are identified using the distance constraint specified. And a consensus of these instances are regarded as the median gene cluster.

#### ***32.4.2.2 Important Issues.*** As addressed by Hoberman and Durand [33], there are two starting points for developing a model: either from a computational point of view with the goal of identifying an efficient algorithm or from a biological point of view with the goal to model biological intuitions as much as possible. The former is interesting theoretically, but for computational biology studies that try to resolve real-life problems, we believe the latter approach is more valuable.

To make more sense of the underlying biology, we summarize the following issues that we believe are important and have not been addressed adequately. Then, we will discuss each of these issues individually.

1. Model evaluation—sensitivity versus specificity
2. Model comparison
3. Model applicability—what should be captured?

Undoubtedly, a more generalized model would have a higher sensitivity than a more restricted model. Unfortunately, in biological applications, high sensitivity without considering specificity amounts to an overwhelming number of wet-lab experiments; hence, it needs be avoided. Therefore, each model should be evaluated for both sensitivity and specificity, which could be carried out in two ways: (i) compare against biologically validated data; (ii) in a less favorable situation, when there is a lack of biological results, comprehensive synthetic data should be used. To date, no comprehensive validations exist in gene cluster detection. And existing approaches attempt to classify the identified gene clusters that could not be validated as the "potential unclassified gene clusters". This conclusion is more convincing in some cases (*e.g.*, HomologyTeams model [32] because a good proportion of the identified gene clusters match biologically validated operons) but is more difficult to judge for some models with a high sensitivity (*e.g.*, median cluster model [9]), because specificity is overlooked when evaluating on both synthetic and real data.

When comparing results, the newly developed models are shown to be superior over existing models by arguing that more gene clusters are found. However, we believe this approach is erroneous. Two important factors need be considered while comparing different models: (i) in addition to sensitivity, specificity needs to be compared; (ii) different methods use different parameter settings, as well as different distance metrics (*e.g.*, physical distance vs. gene distance). The comparable conversion between distance metrics and the parameters are critical before a fair conclusion could be made regarding the sensitivity of two models. Because this conversion sets the sensitivity of both models under comparison to the same level (*i.e.*, the capability to discover a candidate gene cluster) whereas ultimately, the specificity will determine the workload for a biologist doing wet-lab experimentation. As a consequence, ignoring these factors could easily lead to overexaggeration of the capability of the model as well as to misinterpretation of the biological data under study.

Among the models for gene cluster detection, we consider gene teams model as more biologically oriented, which well capture the following evolutionary events: genomic inversions, gene duplications, and gene insertions. Although algorithmically, median gene cluster models can identify these events as well as gene losses, the optimization criteria is more computationally oriented, and the biological meaning of the resulting clusters wanes in multiple genome comparison. Therefore, what has been missed is a model to formally capture gene losses, which is a difficult problem because of the following complications: how to decide if a gene is an insertion in one instance or a loss in another instance; how to differentiate a gene loss event from a gene duplication event if both in-paralogs and out-paralogs are present; and how to deal with genomic inversions, which move the gene loss positions. In addition, the size of instances for a gene cluster is restricted implicitly or explicitly, which rules out the segmental gene loss events in microbial genomes (*e.g.*, the existence of uber-operons [14]).

The first attempt to address several of these issues is given in [70], where an effort for modeling biological intuitions has been made to increase the sensitivity without sacrificing the specificity of the model; gene loss events are modeled formally in line with the gene teams model, and gene cluster consisting of multiple instances that occur in one or multiple genomic sequences is derived without relying on pairwise comparison. While comparing with the existing method, a generic approach is proposed to determine empirically the comparable conversions of distance metrics and choice of parameters. Because the model does not restrict the instances size in a cluster, functional units such as uber-operons are detected automatically.

## 32.5 CONCLUSIONS

In this chapter, we attempt to address the three basic problems in comparative genomics—ortholog assignment, synteny detection, and gene cluster detection. Our goal is to provide an overall picture of how algorithmically these problems are modeled and resolved in the literature rather than provide an exhuastive survey. We refer the reader to subsequent chapters for the detailed discussions of other problems in comparative genomics.

## REFERENCES

1. D.A. Bader, B.M. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *J Comput Biol*, 8(5):483–491, 2001.
2. A.K. Bansal. An automated comparative analysis of 17 complete microbial genomes. *Bioinformatics*, 15(11):900–908, 1999.
3. M. Béal, A. Bergeron, S. Corteel, and M. Raffinot. An algorithmic view of gene teams. *Theor Comput Sci*, 320(2-3):395–418, 2004.
4. S. Bérard, A. Bergeron, C. Chauve, and C. Paul. Perfect sorting by reversals is not always difficult. *IEEE/ACM Trans Comput Biol Bioinform*, 4(1):4–16, 2007.
5. A. Bergeron, C. Chauve, and Y. Gingras. Formal models of gene clusters. In *Computational Techniques and Engineering*. Wiley, New York, 2008.
6. A. Bergeron, S. Corteel, and M. Raffinot. The algorithmic of gene teams. *Lect Notes Bioinform*, 2452:464–476, 2002.
7. M. Blanchette, E.D. Green, W. Miller, and D. Haussler. Reconstructing large regions of an ancestral mammalian genome in silico. *Genome Res*, 14(12):2412–2423, 2004.
8. M. Blanchette, W.J. Kent, C. Riemer, L. Elnitski, A.F.A. Smit, K.M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E.D. Green, D. Haussler, and W. Miller. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Res*, 14(4):708–715, 2004.
9. S. Böcker, K. Jahn, J. Mixtacki, and J. Stoye. Computation of median gene clusters. In M. Vingron and L. Wong, editors, *Research in Computational Molecular Biology*, volume 4955 of *Lecture Notes in Computer*, *Science*, Springer, New York, 2008, pp. 331–345.
10. G. Bourque and P. Pevzner. Genome-scale evolution: Reconstructing gene orders in the ancestral species. *Genome Res*, 12(1):26–36, 2002.
11. G. Bourque, G. Tesler, and P. Pevzner. The convergence of cytogenetics and rearrangement-based models for ancestral genome reconstruction. *Genome Res*, 16(3):311–313, 2006.
12. S. Cannon, A. Kozik, B. Chan, R. Michelmore, and N.D. Young. DiagHunter and genoPix2D: Programs for genomic comparisons, large-scale homology discovery and visualization. *Genome Biol*, 4(10):R68, 2003.
13. S. Cannon and N.D. Young. OrthoParaMap: Distinguishing orthologs from paralogs by integrating comparative genome data and gene phylogenies. *BMC Bioinformatics*, 4:35, 2003.

14. D. Che, G. Li, F. Mao, H. Wu, and Y. Xu. Detecting uber-operons in prokaryotic genomes. *Nucleic Acids Res*, 34(8):2418–2427, 2006.
15. X. Chen, S. Kwong, and M. Li. A compression algorithm for DNA sequences and its applications in genome comparison. *RECOMB '00*, ACM, New York, 2000, p. 107.
16. X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Trans Comput Biol Bioinform*, 2(4):302–315, 2005.
17. J. Chiu, E. Lee, M. Egan, I. Sarkar, G. Coruzzi, and R. DeSalle. OrthologID: Automation of genome-scale ortholog identification within a parsimony framework. *Bioinformatics*, 22(6):699–707, 2006.
18. P. Cliften, P. Sudarsanam, A. Desikan, L. Fulton, B. Fulton, J. Majors, R. Waterston, B.A. Cohen, and M. Johnston. Finding functional features in saccharomyces genomes by phylogenetic footprinting. *Science*, 301(5629):71–76, 2003.
19. A. Darling, B. Mau, F. Blattner, and N. Perna. Mauve: Multiple alignment of conserved genomic sequence with rearrangements. *Genome Res*, 14(7):1394–1403, 2004.
20. C.N. Dewey. Aligning multiple whole genomes with Mercator and MAVID. *Methods Mol Biol*, 395:221–236, 2007.
21. D. Durand and D. Sankoff. Tests for gene clustering. *J Comput Biol*, 10(3-4):453–482, 2003.
22. R. Edgar. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, 32(5):1792–1797, 2004.
23. W.M. Fitch. Distinguishing homologous from analogous proteins. *Syst Zool*, 19(2):99–113, 1970.
24. Z. Fu, X. Chen, V. Vacic, P. Nan, Y. Zhong, and T. Jiang. MSOAR: A high-throughput ortholog assignment system based on genome rearrangement. *J Comput Biol*, 14(9):1160–1175, 2007.
25. Z. Fu and T. Jiang. Clustering of main orthologs for multiple genomes. *J Bioinform Comput Biol*, 6(3):573–584, 2008.
26. W. Fujibuchi, H. Ogata, H. Matsuda, and M. Kanehisa. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and p-quasi grouping. *Nucleic Acids Res*, 28(20):4029–4036, 2000.
27. J.L. Gordon, K.P. Byrne, and K.H. Wolfe. Additions, losses, and rearrangements on the evolutionary route from a reconstructed ancestor to the modern saccharomyces cerevisiae genome. *PLoS Genet*, 5(5):e1000485, 2009.
28. B.J. Haas, A.L. Delcher, J.R. Wortman, and S.L. Salzberg. DAGchainer: A tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18):3643–3646, 2004.
29. S. Hampson, A. McLysaght, B. Gaut, and P. Baldi. LineUp: Statistical detection of chromosomal homology with application to plant comparative genomics. *Genome Res*, 13(5):999–1010, 2003.
30. S.E. Hampson, B.S. Gaut, and P. Baldi. Statistical detection of chromosomal homology using shared-gene density alone. *Bioinformatics*, 21(8):1339–1348, 2005.
31. S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *J ACM*, pp. 178–189, 1995.
32. X. He and M.H. Goldwasser. Identifying conserved gene clusters in the presence of homology families. *J Comput Biol*, 12(6):638–656, 2005.

33. R. Hoberman and D. Durand. The incompatible desiderata of gene cluster properties. In *Comparative Genomics*. Springer-Verlag, Dublin, Ireland, 2005.
34. R. Hoberman, D. Sankoff, and D. Durand. The statistical significance of max-gap clusters. *Lect Notes Comput Sci*, 3388:55–71, 2005.
35. H. Kaplan, R. Shamir, and R.E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J Comput*, pp. 344–351, 1997.
36. M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E.S. Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423(6937):241–254, 2003.
37. S. Kim, J.H. Choi, and J. Yang. Gene teams with relaxed proximity constraint. *Proceedings of the IEEE Computing Systems Bioinformatics Conference*, pp. 44–55, 2005.
38. E.V. Koonin. Orthologs, paralogs, and evolutionary genomics. *Annu Rev Genet*, 39:309–338, 2005.
39. L. Li, C.J. Stoeckert, and D.S. Roos. OrthoMCL: Identification of ortholog groups for eukaryotic genomes. *Genome Res*, 13(9):2178–89, 2003.
40. J. Ma, L. Zhang, B.B. Suh, B.J. Raney, R.C. Burhans, W.J. Kent, M. Blanchette, D. Haussler, and W. Miller. Reconstructing contiguous regions of an ancestral genome. *Genome Res*, 16(12):1557–1565, 2006.
41. S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, 1970.
42. C. Notredame, D.G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1):205–217, 2000.
43. H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Res*, 28(20):4021–4028, 2000.
44. E. Ohlebusch and M. I. Abouelhoda. Chaining algorithms and applications in comparative genomics, In *Handbook of Computational Molecular Biology*. Chapman and Hall, New York, 2006.
45. A. Osterman and R. Overbeek. Missing genes in metabolic pathways: A comparative genomics approach. *Curr Opin Chem Biol*, 7(2):238–251, 2003.
46. M. Ott, J. Zola, S. Aluru, and A. Stamatakis. Large-scale maximum likelihood-based phylogenetic analysis on the IBM bluegene/l. *Proceedings of Supercomputing*, 2007.
47. R. Overbeek, M. Fonstein, M. D'Souza, G.D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *Proc Natl Acad Sci U S A*, 96(6):2896–2901, 1999.
48. L. Parida. Statistical significance of large gene clusters. *J Comput Biol*, 14(9):1145–1159, 2007.
49. S. Pasek, A. Bergeron, J.L. Risler, A. Louis, E. Ollivier, and M. Raffinot. Identification of genomic features using microsyntenies of domains: Domain teams. *Genome Res*, 15(6):867–874, 2005.
50. E. Passarge, B. Horsthemke, and R.A. Farber. Incorrect use of the term synteny. *Nat Genet*, 23(4):387, 1999.
51. P. Pevzner and G. Tesler. Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes. *Genome Res*, 13(1):37–45, 2003.

52. N. Raghupathy and D. Durand. Individual gene cluster statistics in noisy maps. In *Comparative Genomics*, volume 3678 of *Lecture Notes in Computer Science*, Springer, New York, 2005.

53. N. Raghupathy, R. Hoberman, and D. Durand. Two plus two does not equal three: Statistical tests for multiple genome comparison. *J Bioinform Comput Biol*, 6(1):1–22, 2008.

54. S. Rahmann and G.W. Klau. Integer linear programs for discovering approximate gene clusters. *WABI*, pp. 298–309, 2006.

55. M. Remm, C.E. Storm, and E.L. Sonnhammer. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J Mol Biol*, 314(5):1041–1052, 2001.

56. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

57. D. Sankoff and L. Haque. Power boosts for cluster tests. In *Comparative Genomics*, volume 3678 of *Lecture Notes of Computer Science*, Springer, New York, 2005, pp. 121–130.

58. A.C. Siepel. An algorithm to enumerate sorting reversals for signed permutations. *J Comput Biol*, 10(3-4):575–597, 2003.

59. C. Simillion, K. Janssens, L. Sterck, and Y. Van de Peer. i-ADHoRe 2.0: An improved tool to detect degenerated genomic homology using genomic profiles. *Bioinformatics*, 24(1):127–128, 2008.

60. C. Simillion, K. Vandepoele, Y. Saeys, and Y. Van de Peer. Building genomic profiles for uncovering segmental homology in the twilight zone. *Genome Res*, 14(6):1095–1106, 2004.

61. B. Snel, P. Bork, and M. Huynen. The identification of functional modules from the genomic association of genes. *Proc Natl Acad Sci*, 99(9):5890–5895, 2002.

62. C.E.V. Storm and E.L.L. Sonnhammer. Automated ortholog inference from phylogenetic trees and calculation of orthology reliability. *Bioinformatics*, 18(1):92–99, 2002.

63. R.L. Tatusov, M.Y. Galperin, D.A. Natale, and E.V. Koonin. The COG database: A tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Res*, 28(1):33–36, 2000.

64. T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26:2000, 2000.

65. K. Vandepoele, Y. Saeys, C. Simillion, J. Raes, and Y. Van De Peer. The automatic detection of homologous regions (ADHoRe) and its application to microcolinearity between arabidopsis and rice. *Genome Res*, 12(11):1792–1801, 2002.

66. K. Vandepoele, C. Simillion, and Y. Van de Peer. Detecting the undetectable: Uncovering duplicated segments in arabidopsis by comparison with rice. *Trends Genet*, 18(12):606–608, 2002.

67. A. Vashist, C.A Kulikowski, and I. Muchnik. Ortholog clustering on a multipartite graph. *IEEE/ACM Trans Comput Biol Bioinform*, 4(1):17–27, 2007.

68. S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

69. Q. Yang and S. Sze. Large-scale analysis of gene clustering in bacteria. *Genome Res*, 18(6):949–956, 2008.

70. X. Yang and S. Aluru. A unified model for multi-genome synteny and gene cluster inference. Technical Report, Iowa State university, Ames IA, 2009. http://archives.ece.iastate.edu/archive/00000496/.

71. C. Zheng, Q. Zhu, and D. Sankoff. Removing noise and ambiguities from comparative maps in rearrangement analysis. *IEEE/ACM Trans Comput Biol Bioinform*, 4(4):515–522, 2007.

72. H. Zhu, D. Kim, J. Baek, H. Choi, L. Ellis, H. Küester, W. McCombie, H. Peng, and D. Cook. Syntenic relationships between medicago truncatula and arabidopsis reveal extensive divergence of genome organization. *Plant Physiol*, 131(3):1018–1026, 2003.

73. Q. Zhu, Z. Adam, V. Choi, and D. Sankoff. Generalized gene adjacencies, graph bandwidth and clusters in yeast evolution. *ISBRA*, 2008, pp. 134–145.

74. J. Zola, X. Yang, S. Rospondek, and S. Aluru. Parallel T-Coffee: A parallel multiple sequence aligner. *Proceedings of the ISCA PDCS*, 2007, pp. 248–253.

# 33

# ADVANCES IN GENOME REARRANGEMENT ALGORITHMS

Masud Hasan and M. Sohel Rahman

## 33.1 INTRODUCTION

One of the goals of the scientists is to explain better the evolutionary history of a set of species. Interestingly although the organizations of the molecules of different species differ dramatically, the content of the DNA molecules from one species to another are believed to be often similar. Genome rearrangements refer to the mutations that affect this organization. Now, the role of computer scientists (or bioinformaticians), in this regard, lies basically in the efficient formulation of the evolutionary events to a combinatorial problem and then to provide an efficient solution for it. More specifically, a computer scientist would use mathematical model and combinatorial tools to reconstruct rearrangement scenarios to explain better the evolutionary history of a set of species.

Now, every study of genome rearrangement involves solving a combinatorial problem of finding a series of rearrangements that transform a genome into another. In the late 1980s, Palmer and Herbon [70] found that the number of such operations needed to transform the gene order of one genome into the other could be used as a measure of the evolutionary distance between two species. The classic examples of cabbage transforming into a turnip (Figure 33.1) and/or a mouse transforming into a human (Figure 33.2) are cited in almost all textbooks related to bioinformatics. Notably, in the latter example, the focus is only on the *X* chromosome, because

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

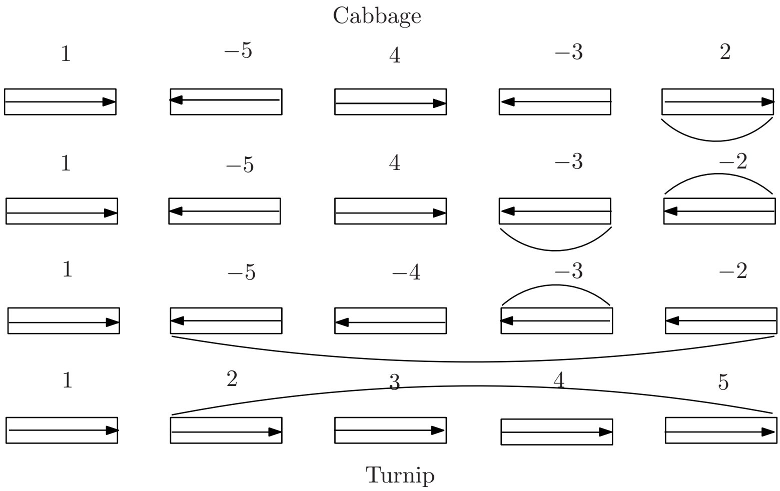


**Figure 33.1** Transformation of a cabbage into a turnip.

according to Ohno's law, the gene content of *X* chromosomes barely has changed throughout mammalian evolution, although, the order of genes therein has been disrupted several times. One of the exciting and perhaps surprising findings of the scientists is that, in some ways, the human genome is just the mouse genome cut into about 300 large genomic fragments, called the synteny blocks, that have been pasted together in a different order. And, although there are about 80 million years of evolutionary distance between the two, this amounts to only about 140–150 operations of rearrangements!

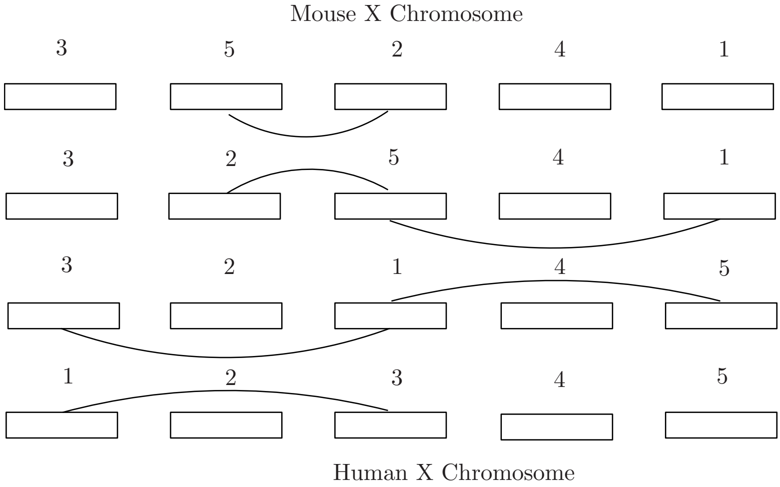


**Figure 33.2** Transformation of the mouse gene order into the human gene order on the *X* chromosome (only the five longest synteny blocks are shown here).

One question is what kind of genome rearrangement events (also called operations) take place in nature? If we consider operations on a single chromosome, then the possible operations include deletions (a certain part is lost), insertions (a part is added), duplications, reversals or inversions (a part is reversed), and transpositions (two parts change places). Among these transposition is believed to be the most rare because it requires three points of leverage along the chromosome. Now, how do these operations take place? If two regions along a chromosome are very similar, then they might hybridize just like two different strands of the double helix. Once they are attached, a loop forms. This loop might be discarded (deletion), or its direction might switch (reversal). Operations on two chromosomes include translocation (two chromosomes swap their "tails"), fusion (two chromosomes merge), and fission (one chromosome splits into two chromosomes). Scientific studies show that for uni-chromosomal genomes, inversions (reversals) are the dominant rearrangement event, and for multichromosomal genomes, reversals, transpositions, and translocations are common rearrangement events. Biologists are interested in the most parsimonious evolutionary scenario, that is, the scenario involving the smallest number of operations. It is important to note that, although there is no guarantee that this scenario represents an actual evolutionary sequence, it gives us a lower bound on the number of rearrangements that have occurred and indicates the similarity between two species in some way. Another notable and perhaps more interesting point from an informatics point of view is that even for the small number of synteny blocks shown, it is not so easy to verify that the three evolutionary events in Figure 33.2 represent the shortest series of reversals transforming the mouse gene order into the human gene order on the *X* chromosome. This makes the study of genome rearrangement interesting and challenging for the computer scientists from the complexity as well as algorithmic points of view.

The analysis of genome rearrangements in molecular biology was pioneered in the late 1930s by Dobzhansky and Sturtevant, who published a milestone paper presenting a rearrangement scenario with 17 inversions for the species of *Drosophila* fruit fly [29]. Since then and with the advent of large-scale DNA mapping and sequencing, the number of biological problems related to genome rearrangements has been growing rapidly in different areas including the evolution of Lobelia fervens [59], chloroplast [72], and mitochondrial genomes [74], virology [62], and *Drosophila* genetics [83]. However, the first computer science results allowing a biologist to analyze gene rearrangements appeared in the literature no earlier than the early 1990s. The first such result was an algorithm by Kececioglu and Sankoff [58] for reversal distance with a guaranteed error bound two. The abovementioned paper raised a spectrum of open problems motivated by genome rearrangements that opened a new avenue of research in this regard. However, as it turned out, most problems related to genome arrangements are nondeterministic polynomial (NP)-hard; some versions of the problems are still open from the complexity point of view. As a result, the main thrust of the research has been directed toward the approximation algorithms of the related problems. In this chapter, we make an effort to present the state of the art of the genome rearrangements algorithms in the computer science literature.

The rest of the chapter is organized as follows. We start with some preliminary notations and definitions in Section 33.2. The problem of sorting by reversals, both for signed and unsigned versions, is studied in Section 33.3. Section 33.4 focuses on the problem of sorting by transpositions. We discuss different other rearrangement operations in Section 33.5. Then, in Section 33.6, we focus on applying more than one operation simultaneously. Finally, we conclude the chapter by discussing some future research directions in Section 33.7.

## 33.2 PRELIMINARIES

A permutation of a sequence of $n$ numbers is just a reordering of that sequence. In what follows, we always will use permutations of consecutive integers. For example, $(2, 1, 3, 4, 5)$ is a permutation of $(1, 2, 3, 4, 5)$. Notably, the latter sequence is termed as the identity permutation. As has been mentioned before, synteny blocks are genomic fragments and play important roles in comparative genomics analysis. In the rearrangement of synteny blocks in a genome, the molecules within a synteny block do not change the order. For this reason, a synteny block is considered as a single element and the whole genome is a permutation of a certain length. For example, the order of synteny blocks on the $X$ chromosome in humans is represented in Figure 33.2 by $(1, 2, 3, 4, 5)$, whereas the ordering in mice is $(3, 5, 2, 4, 1)$. Now, the problem of finding a sequence of rearrangement operations to transform one genome to another reduces to transforming one permutation to another. The problem can be simplified further to finding a sequence of rearrangement operations to transform one permutation to the identity permutation. Moreover, recall that, the biologists are interested in finding the shortest such sequence.

As has been mentioned already, most problems related to the genome arrangements and hence to sorting permutations, have designated to be hard to solve. In most problems discussed in this chapter, efficient polynomial algorithms are still unknown and are unlikely ever to be found. Therefore, we are interested in approximation algorithms for the problems at hand. Note carefully that approximation algorithms are only relevant to problems that have a numerical objective function (*i.e.*, are maximization/minimization problems). For a maximization problem, the *approximation ratio*, or *performance guarantee*, denoted as $\rho$, of algorithm $\mathcal{A}$ is defined as the maximum ratio of $\mathcal{A}(\pi)$ and $\text{OPT}(\pi)$ over all input $\pi$, that is, as

$$\rho \leq \max_{\pi} \frac{\text{OPT}(\pi)}{\mathcal{A}(\pi)}$$

where $\mathcal{A}(\pi)$ and $\text{OPT}(\pi)$ refer to the solutions produced by the Algorithm $\mathcal{A}$ and the correct (optimal) solution of the problem, respectively. For a minimization problem it is defined as follows:

$$\rho \leq \max_{\pi} \frac{\mathcal{A}(\pi)}{\text{OPT}(\pi)}$$

which means that the performance of $\mathcal{A}$ is measured by how much $\rho$ can become close to one. Note that in our case the problem is a minimization problem, and we will follow the second equation. It should be clear that an approximation algorithm gives a worst-case scenario of just how far off an algorithms output can be from a hypothetical perfect algorithm. The reason we use the term "hypothetical perfect" is because, in reality, such a case may not even develop. Technically speaking, an approximation algorithm would return a suboptimal (incorrect) output for some input. The approximation ratio gives us an idea of just how incorrect the algorithm can be. Clearly, the goal is to design approximation algorithms with better performance guarantees. One final remark is that the overall running time of the algorithm must remain polynomial for the approximation algorithm to be useful. In other words, an exponential-time approximation algorithm is not acceptable irrespective of its approximation ratio.

## 33.3 SORTING BY REVERSALS

As has been mentioned already, scientific studies show that for unichromosomal genomes, reversals are the dominant rearrangement operation. Therefore, we start this chapter with the problem of sorting, allowing only the reversal operations. This particular problem has been popularly referred to in the literature as the problem of *sorting by reversals*. The section outline is as follows. First we define the problem more formally. Then we discuss how to design an approximation algorithm for sorting by reversals followed by some techniques to improve the approximation ratio. We further discuss the relation with the signed version of the problem (*i.e.*, when each element of the permutation has a sign "+" or "−", which changes with the element comes under a reversal). Finally, we give other recent results on the improvement on the approximation ratio and running time.

We begin by defining some notations and terminologies used throughout this section and the subsequent sections. A genome can be represented by a permutation $\pi = [\pi_1\ \pi_2\ \dots\ \pi_n]$ of $n$ distinct elements, where $1 \leq \pi_i \leq n$ for $1 \leq i \leq n$. A *reversal* $\rho(i, j)$, for some $1 \leq i < j \leq n$, applied to $\pi$ reverses the elements $\pi_i\ \dots\ \pi_{j-1}$ and thus transforms $\pi$ into permutation $\pi \times \rho = [\pi_1\ \dots\ \pi_{i-1}\ \pi_{j-1}\ \dots\ \pi_i\ \pi_j\ \dots\ \pi_n]$. For example, if $\pi = 2\ 3\ \underset{\longleftrightarrow}{1\ 7\ 4}\ 5\ 6$, then $\pi \times \rho(3, 6) = 2\ 3\ 4\ 7\ 1\ 5\ 6$.

An *identity permutation* $I$ is a permutation such that $\pi_i = i$ for $0 \leq i \leq n + 1$. The *reversal distance* $d(\pi)$ between $\pi$ and $I$ is the minimum number of reversals such that $\pi \times \rho_1 \times \rho_2 \times\ \dots\ \times \rho_{d(\pi)} = I$. The problem of *sorting by reversals* is to find a shortest sequence of reversals that transforms a permutation $\pi$ into identity permutation $I$ (*i.e.*, finding the distance $d(\pi)$). For example, $\pi = 2\ 3\ 1\ 7\ 4\ 5\ 6$ can be sorted by minimum $d(\pi) = 3$ reversals: $2\ 3\ 1\ 7\ \underset{\longleftrightarrow}{4\ 5\ 6} \rightarrow \underset{\longleftrightarrow}{2\ 3}\ 1\ 7\ 6\ 5\ 4 \rightarrow 1\ 2\ 3\ \underset{\longleftrightarrow}{7\ 6\ 5}\ 4 \rightarrow 1\ 2\ 3\ 4\ 5\ 6\ 7$.

In what follows, as we have mentioned before, we will be focusing on unsigned permutations. And, because the sorting by reversal problem for unsigned permutations are proven to be NP-hard [19], we will concentrate on approximation algorithms. We will talk about the signed version of the problem in a later section. Before going deep into the discussion of approximation algorithms for the sorting by reversal problem for the unsigned permutation, we would like to discuss a bit of the motivation behind studying the unsigned version of the problem. Biologists derive gene orders either by sequencing entire genomes or by using comparative physical mapping. Sequencing provides information about the directions of genes and allows one to represent a genome by a signed permutation. However, sequencing of entire genomes is still expensive, and most currently available experimental data on gene orders are based on comparative physical maps. Physical maps usually do not provide information about the directions of genes and therefore lead to the representation of a genome as an unsigned permutation.

### 33.3.1 Approaches to Approximation Algorithms

In this section, we discuss different approaches to solve the sorting by reversal problem. We start with the so-called *breakpoint* model, which in fact, was introduced in the very first computational studies of genome rearrangements [69, 82]. In particular, the authors of the pioneering papers noticed some correlations between the reversal distance and the number of breakpoints, which interestingly enough, were discussed implicitly by Sturtevant and Dobzhansky [75], almost 70 years ago! We define the notion of a breakpoint subsequently.

A *breakpoint* in $\pi$ is a position $i$ of a permutation $\pi$ such that $|\pi_i - \pi_{i-1}| \neq 1$. The number of breakpoints of permutation $\pi$ is denoted by $b(\pi)$. Therefore, $b(\pi) \geq 1$ if and only if the permutation $\pi$ is not $I$. The most important concept of designing an approximation algorithm for sorting by reversals (in fact, sorting by any operations) is based on efficiently removing break points from $\pi$.

Designing a good approximation algorithm depends on finding a good lower bound for an optimal algorithm as well as giving an algorithm with a good upper bound. We first present a classic two-approximation algorithm that illustrates the basic approach of finding a lower bound and an upper bound for finding an approximation algorithm for sorting by reversals. This algorithm is from Kececioglu and Sankoff [58].

#### *33.3.1.1 A Classic 2-approximation Algorithm.*

It is convenient to add two extra elements $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ at the beginning and end of $\pi$, respectively, and thus considering $\pi = [\pi_0, \pi_1\ \pi_2\ \ldots\ \pi_n, \pi_{n+1}]$. To transform $\pi$ to the identity permutation, the nonconsecutive elements $\pi_i$ and $\pi_{i+1}$ forming a breakpoint must be removed. The sorting by reversals is the process of eliminating such breakpoints. The main observation is that a reversal can eliminate at most two breakpoints: one on the left end and another on the right end of the reversal. It implies that an optimal algorithm must use at least $\frac{b(\pi)}{2}$ reversals, giving a lower bound of $d(\pi) \geq \frac{b(\pi)}{2}$ for sorting by reversals.

**Lemma 33.1** $d(\pi) \geq \frac{b(\pi)}{2}$.

Now we present the idea of an algorithm that will use at most $b(\pi)$ reversals, thus giving an upper bound of $d(\pi) \leq b(\pi)$. It needs to partition $\pi$ into *increasing* and *decreasing strips*. A *strip* of $\pi$ is a maximal subsequence of $\pi$ without any breakpoint. A strip of one element is both increasing and decreasing, except for $\pi_0$ and $\pi_{n+1}$, which are always increasing. For example, $\pi = 0\ 2\ 3\ 1\ 7\ 4\ 5\ 6\ 7$ has six strips: 0 | 2 3 | 1 | 7 | 4 5 6 | 8, where | 0 |, | 2 3 |, | 4 5 6 |, and | 8 | are the increasing strips, and the remaining strips are both increasing and decreasing.

From the lower bound in Lemma 33.1, a reversal can add or remove no more than two breakpoints. Define an *i-reversal*, $i \in \{0, 1, 2\}$, as the reversal that removes $i$ breakpoints. Now the algorithm removes breakpoints by reversals and merge strips into a single increasing strip. To guarantee an approximation ratio of 2, it always should happen that we can find a reversal that removes at least one breakpoint, at least on *average*. Kececioglu and Sankoff [58] proved that this is possible by the following lemma.

**Lemma 33.2** *If $\pi$ has a decreasing strip, then it is always possible to find a one- or two-reversal on $\pi$. If every possible reversal transforms $\pi$ without any decreasing strip, then among all those possible reversals, there is at least one two-reversal.*

So, a greedy algorithm for sorting by reversals is to apply reversals in *rounds*. In each round, apply a reversal that removes maximum break points as long as we are left with $\pi$ with no decreasing strip, for which we apply a zero-reversal to convert an increasing strip to a decreasing strip. Then start a new round.

By Lemma 33.2, we know that at the end of each round, we applied a two-reversal, which can *amortize* the zero-reversal at the beginning of the next round. So, as a whole, each reversal removes at least one breakpoint on average. Therefore, we need no more than $b(\pi)$ reversals to sort $\pi$.

**Lemma 33.3** $d(\pi) \leq b(\pi)$.

Lemmas 33.1 and 33.3 give an approximation ratio $\rho \leq \frac{b(\pi)}{\frac{b(\pi)}{2}} = 2$. It is worth mentioning that in this algorithm, finding a reversal that removes maximum breakpoint can be done in polynomial time once a suitable representation of $\pi$ is assumed.

**Theorem 33.1** *A polynomial time two-approximation algorithm exists for sorting by reversals.*

We remark that although this algorithm was the first of this kind, it gives a basic concept of designing subsequent more efficient approximation algorithms.

***33.3.1.2 Improving by Cycle Decomposition Graph.*** Although there has been much research work in the literature on the breakpoint model, it turns out that the estimate of reversal distance in terms of breakpoints is very inaccurate. In 1993, Bafna and Pevzner in their pioneering paper [8, 10] showed that another parameter (size of a maximum cycle decomposition of the breakpoint graph) estimates reversal distance with a much greater accuracy. In this section, we focus on the cycle decomposition graph model to solve the problem.

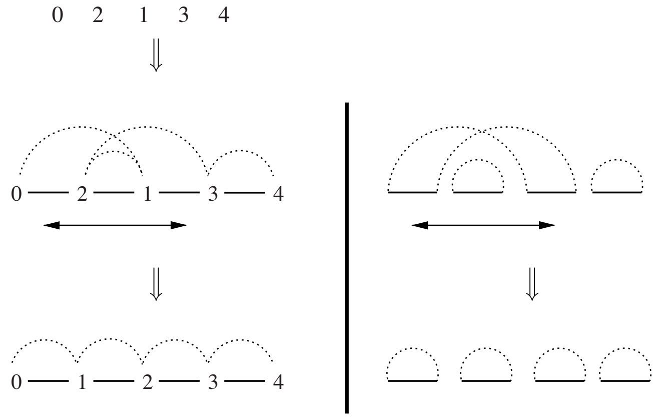


**Figure 33.3** A permutation $\pi$ and its cycle decomposition graph $G(\pi)$. A reversal $\rho(2, 3)$ applied on $G(\pi)$. The right-hand side shows a clearer picture of the changes in $G(\pi)$ that happen as a result of the reversal.

The *breakpoint graph*, also called *cycle decomposition graph*, $G(\pi)$ of $\pi$ is an undirected multigraph whose $n+2$ vertices are $\pi_i$, for $0 \leq i \leq n+1$. $G(\pi)$ has $2(n+1)$ edges, and they are of two types: *gray* and *black*. For each $0 \leq i \leq n$, the vertices $\pi_i$ and $\pi_{i+1}$ are joined by a black edge. For $0 \leq i, j \leq n+1$, there is a gray edge between $\pi_i$ and $\pi_j$ if $\pi_i = \pi_j + 1$. For example, Figure 33.3 shows the breakpoint graph of the permutation 0 2 1 3 4.

An *alternating cycle* in $G(\pi)$ is a cycle of size at least two in which the edges are alternate colors. An *l-cycle* means an alternating cycle with $l$ black edges. The graph $G(\pi)$ can be decomposed completely into edge-disjoint alternating cycles [23, 20, 63]. However, there may be many different such *cycle decompositions*. The maximum number of cycles in any cycle decomposition of $G(\pi)$ is denoted by $c(\pi)$.

At first, it seems that no or very little connection exists between cycle decomposition of $G(\pi)$ and sorting by reversals. However, there is a strong connection between breakpoint and the value of $l$ in an $l$-cycle. Observe that an $l$-cycle involves two consecutive elements of $\pi$ that have no breakpoint in it. In fact, $\pi$ is the identity permutation if and only if $G(\pi)$ does not have any $l$-cycle for $l \geq 2$, and in that case, $\pi$ has exactly $n+1$ $l$-cycles. That means the sorting by reversals is the process of increasing the number of $l$-cycles in $G(\pi)$, and efficiently increasing this number by decomposing larger cycles can lead to better approximation ratio. (See Figure 33.3.) For example, the lower bound of $d(\pi)$ can be improved using this idea as follows. Bafna and Pevzner [8, 10] proved that any optimal algorithm cannot increase the number of cycles by more than one, which gives a stronger lower bound of $d(\pi) \geq b(\pi) - c(\pi)$. This lower bound is occasionally better than $b(\pi)/2$ because an $l$-cycle means adjacency between two elements, and the black edges in every other $l$-cycle, for $l \geq 2$, means a breakpoint. Because there can be at most $b(\pi)$ black edges in all $l$-cycles, for $l \geq 2$, the number of such $l$-cycles can be at most $b(\pi)/2$.

Now for an improved upper bound, if we can start with a maximum cycle decomposition of $G(\pi)$, then our task of increasing the number of $l$-cycle becomes easy. But finding a maximum cycle decomposition is not easy; in fact, it is proven to be NP-hard [19]. So people started decomposing $G(\pi)$ into cycles in many different ways and estimating the number of larger cycles in a maximum cycle decomposition of $G(\pi)$. For example, Bafna and Pevzner [8, 10] found a decomposition that has at least a certain number of four-cycles. Using that, they further improved the lower bound to $d(\pi) \geq \frac{2}{3}b(\pi) - \frac{1}{4}c_4(\pi)$, where $c_4(\pi)$ is the number of four-cycles in a maximum cycle decomposition. And they give an algorithm that uses at most $d(\pi) \leq b(\pi) - \frac{1}{4}c_4(\pi)$ reversals. Combining these two gives an improved approximation ratio of $\rho \leq \frac{7}{4}$.

Using a similar idea, Christie [23] proved a lower bound of $d(\pi) \geq \frac{2}{3}b(\pi) - \frac{1}{3}c_2(\pi)$, where $c_2(\pi)$ is the number of two-cycles in their cycle decomposition and gave an algorithm that uses at most $b(\pi) - \frac{1}{2}c_2(\pi)$ reversals. Thus, their algorithm gives an approximation ratio of $\frac{3}{2}$.

Finally, the best approximation ratio so far comes from Berman *et al.* [15] with ratio 1.375, which they achieve by developing a new approximation algorithm for maximum cycle decomposition.

***33.3.1.3 How Much Can One Improve?*** Not much, at least on complexity issues! Caprara already proved that sorting by reversal problem is NP-hard [19]. Their proof uses, once again, the concept of cycle decomposition of $G(\pi)$. They proved that the maximum cycle decomposition of $G(\pi)$ is equivalent to maximum cycle decomposition of an Eulerian graph, which is proven to be NP-hard in [54]. Then they reduce the problem of sorting by reversal to maximum cycle decomposition of $G(\pi)$, thus proving the former problem NP-hard, too.

That is not the end of bad news on improving on sorting by reversals. Berman and Karpinski [16] proved that for sorting by reversals, a constant factor approximation algorithm is the best one can think of. They proved that it cannot be approximated better than 1.008 times the optimal, thus excluding the possibility of a polynomial time approximation scheme.[1]

**Open Problem 33.1** *For the approximation ratio, there is a wide gap between the best known lower bound of 1.008 and the best known upper bound of 1.375. Is it possible to close this gap by finding an approximation ratio better than 1.375 or by improving the inapproximability result of 1.008?*

### 33.3.2 Signed Permutations

In the previous subsection, we have represented a genome by unsigned permutations. However, in reality, genes and synteny blocks have directionality. This directionality

[1] A polynomial time approximation scheme can approximate arbitrary close to the optimal, but in expense of a higher running time.

basically reflects whether they reside on the direct strand or the reverse complement strand of the DNA. Therefore, the synteny block order in an organism is represented better by a signed permutation instead of an unsigned one. Notably, the motivation presented in the previous section for unsigned version still remains valid. Just to remind the readers, because of the high cost of sequencing experimentations, gene orders derivation is done mostly using comparative physical mapping, which usually does not provide information about the directions of genes and therefore leads to the representation of a genome as an unsigned permutation $\pi$. Now, biologists make an effort to derive a signed permutation from this representation by assigning a positive (negative) sign to increasing (decreasing) strips[2] of $\pi$. And in fact, the result of Hannenhalli and Pevzner [45] that "reversals do not cut long strips" do provide a theoretical substantiation for such a procedure in the case of long strips. Admittedly, however, for two-strips, this procedure might fail to find an optimal rearrangement scenario. Hannenhalli and Pevzner [45] pointed to a biological example for which this procedure fails and described an algorithm fixing this problem. Nevertheless, the fascinating and intriguing difference in the complexity status of the signed version with the unsigned one along with its biological significance makes the problem very interesting to study.

In a signed permutation $\pi'$, each element $\pi'_i$ has a sign either "+" or "−". After a reversal, the sign of the elements in the subsequence that has been reversed is changed. At first sight, it seems that, involving an extra sign parameter, sorting signed permutation by reversals would be difficult. However, it is interesting that sorting a signed permutation by reversals is easier to handle than unsigned permutations; in fact, it can be handled by the unsigned version. A signed permutation $\pi'$ is converted to an unsigned permutation $\pi$as follows: each $+x$ element is replaced with $2x - 1$ and $2x$, and each $-x$ element is replaced $2x$ and $2x - 1$. Then the cycle decomposition graph $G(\pi')$ of $\pi'$ is defined as the cycle decomposition graph $G(\pi)$ of $\pi$ with its one-cycles deleted. For example, see Figure 33.4.

Observe that in $G(\pi')$, each vertex has degree two. Thus, $G(\pi')$ already is decomposed into disjoin cycles, and it is the maximum decomposition. This is the fact that makes sorting a signed permutation easier. From this decomposition, an optimal sequence of reversals can be found in polynomial time that sorts $\pi'$ into an identity permutation. Hannenhalli and Pevzner [44] were the first to give a polynomial time alorithm for this problem. Their algorithm runs in $O(n^4)$ time. But before that, when the complexity was unknown, Kececioglu and Sankoff [58] conjectured that the problem is NP-hard and gave a two-approximation algorithm. Later, Bafna and Pevzner [8, 10] improved the ratio to 1.5 by using the breakpoint graph.

***33.3.2.1 Improved Running Time and Simpler Algorithms.*** Most improvements and simplifications of the algorithms for sorting by reversals are for signed permutations. Once Henehalli and Pevzner found an optimal polynomial time solution for signed permutation, people started to make it faster and simpler. The algorithm of Hannenhalli and Pevzner takes $O(n^4)$ time. Proving that Kececieglu

[2] Recall that a strip of $\pi$ is a maximal subsequence of $\pi$ without any breakpoint. A strip of one, two, and more than two elements are called, respectively, a singleton, a 2-strip, and a long strip.

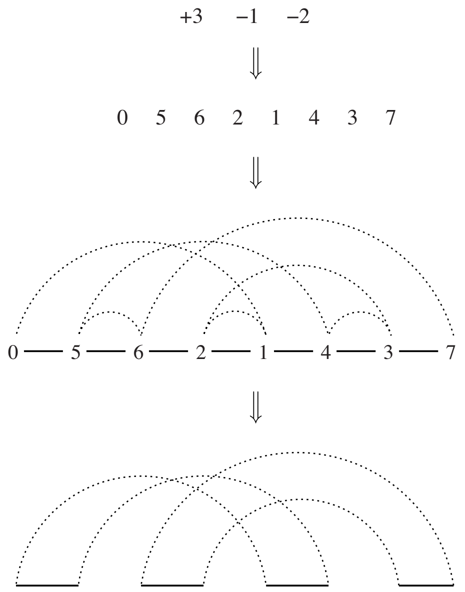


**Figure 33.4** Transforming a signed permutation to an unsigned one and creating a corresponding cycle decomposition graph.

and Sankoff's conjecture in negative was good enough to leave an algorithm with such a higher running time. Berman and Henehalli [14] implemented their algorithm in $O(n^2\alpha)$ time, where $\alpha$ is the inverse Ackerman function. Kaplan *et al.* [55] simplified the algorithm and improved the runing time to $O(n^2)$. Since then, it was a challenge to design a subquadratic algorithm. Bader *et al.* [7] designed a linear time algorithm for computing $d(\pi)$ without giving the actual sequence of reversals of that length. Kaplan and Verbin [56] broke the quadratic bound by giving a randomized $O(n^{\frac{1}{3}}\sqrt{\log n})$ time algorithm. Then Tannier *et al.* [77] achieved a deterministic $O(n^{\frac{1}{3}}\sqrt{\log n})$ time algorithm. Very recently, Swenson *et al.* [76] gave an algorithm that runs in $O(n \log n + kn)$ time, where $k$ is mostly a constant but $\theta(n)$ is in worst case. Despite all these efforts, an $O(n \log n)$ algorithm remains ellusive.

**Open Problem 33.2** *Devise an $O(n \log n)$ algorithm for sorting signed permutation by reversals.*

For now, that is all for the "pure form" of sorting by reversals. We will discuss the variants of reversals in Section 33.5.

## 33.4 SORTING BY TRANSPOSITIONS

In the previous section, we have focused on the most common rearrangement operation, namely the reversals. In this section, we focus on transposition. As we have mentioned before, transposition is believed to be the most rare because it requires three points of leverage along the chromosome. However, in case of highly rearranged genomes (*e.g.*, herpes viruses or plant mitochondrial DNA) transpositions of a long fragment along with reversals seem to be common. To be more specific, analysis of genomes of the Epstein-Barr Virus (EBV) and herper simplex virus (HSV-1) reveals that evolution of these herpes viruses involve several inversions and transpositions of large fragments; in particular, an analogue of the gene UL52-BSLF1 (required for DNA replication) in a common herpes virus precursor "jumped" from one location in the genome to another. This later event of jumping in fact refers to the transposition. As a first approximation, transpositions in genome rearrangements can be modeled in a straightforward manner by the problem of *sorting by transpositions*, which is the subject matter of this section.

A *transposition* $\tau = \tau(i, j, k)$, for some $1 \leq i < j \leq n+1$ and some $1 \leq k \leq n+1$ such that $k \notin [i, j]$ cuts the elements $\pi_i \ \ldots \ \pi_{j-1}$ and pastes between $\pi_{k-1}$ and $\pi_k$ and thus transforms $\pi$ into permutation $\pi \times \tau = [\pi_0 \ldots \pi_{i-1}\, \pi_j \ldots \pi_{k-1}\, \pi_i \ldots \pi_{j-1}\, \pi_k \ldots \pi_{n+1}]$. In other words, a *transposition* $\tau = \tau(i, j, k)$ swaps the two consecutive blocks $\boxed{\pi_i \ \ldots \ \pi_{j-1}}$ and $\boxed{\pi_j \ \ldots \ \pi_{k-1}}$. For example, if $\pi = 0\ 2\ 1\ 3\ \boxed{7}\ \boxed{4\ 5\ 6}\ 8$, then $\pi \times \tau(4, 5, 8) = 0\ 2\ 1\ 3\ \boxed{4\ 5\ 6}\ \boxed{7}\ 8$.

The *transposition distance* $d(\pi)$ between $\pi$ and $I$ is the minimum number of reversals such that $\pi \times \tau_1 \times \tau_2 \times \cdots \times \tau_{d(\pi)} = I$. The problem of *sorting by transposition* is to find a shortest sequence of reversals that transforms a permutation $\pi$ into identity permutation $I$ (*i.e.*, finding the distance $d(\pi)$). For example, $\pi = 0\ 2\ 1\ 3\ 7\ 4\ 5\ 6\ 8$ can be sorted by the following minimum $d(\pi) =$ two transpositions: $0\ 2\ 1\ 3\ \boxed{7}\ \boxed{4\ 5\ 6}\ 8 \rightarrow 0\ \boxed{2}\ \boxed{1}\ 3\ 4\ 5\ 6\ 7\ 8 \rightarrow 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8$.

### 33.4.1 Approximation Results

Once again, similar to sorting by reversals, a good approximation algorithm tries to remove breakpoints in $\pi$ as quickly as possible. A transposition can remove at most three breakpoints. For instance, in the previous example, the first transposition removes three breakpoints. So, a lower bound for sorting by transpositions is $d(\pi) \geq \frac{b(\pi)}{3}$. With this lower bound, any algorithm that can remove at least one breakpoint in each step will gurantee an approximation ratio of 3, and such an algorithm is not difficult to design.

But incorporating a cycle decomposition graph $G(\pi)$ of $\pi$ can improve greatly the approximation ratio. It is not always possible that an optimal algorithm will remove three breakpoints at each steps. So, the earlier lower bound is not tight. Remember that for the identity permutation $\pi = I$, the number of cycles is $n+1$, where all cycles are $l$-cycle. Bafna and Pevzner [9, 11] proved that if the number of cycles with an odd number of black edges (called the *odd* cycles) in $G(\pi)$ is $c_{\text{odd}}(\pi)$, then a transposition can increase the number of cycles in $G(\pi)$ by at most two. So an improved lower bound is $d(\pi) \geq \frac{n+1-c_{\text{odd}}(\pi)}{2}$. They also give an algorithm that can increase, at least on average, one odd cycle in each step, thus giving an upper bound of $d(\pi) \leq n+1-c_{\text{odd}}(\pi)$. This gives an approximation ratio of 3. In the same paper [9, 11], they improve the approximation ratio to 1.75 and then to 1.5. The ratio 1.5 was the best one for about 10 years until Elias and Hartman [33] came up with a ratio of 1.375 with an $O(n^2)$ time algorithm.

### 33.4.2 Improved Running Time and Simpler Algorithms

In parallel, people kept finding simpler and faster algorithms for sorting by transpositions, sometimes even sacrificing approximation ratios. The 1.5 approximation algorithm of Bafna and Pevzner [9, 11] runs in $O(n^2)$ time, but it is somewhat difficult. Christei [24] came up with a simpler $O(n^4)$ time 1.5 approximation algorithm. An $O(n^3)$ implementation of this algorithm, along with heuristics that improve its performance, were given by Walter *et al.* [81]. Eriksson *et al.* [35] provided an algorithm that sorts any given permutation on $n$ elements by at most $2n/3$ transpositions but has no approximation guarantee. Hartman [46] further simplified the algorithm and improved the runing time to $O(n^2)$, and then Hartman and Shamir [47] further improved the runing time to $n^{3/2}\sqrt{\log n}$ by using a splay tree. But despite all these efforts, an $O(n \log n)$ algorithm remained ellusive, until recently, when Feng and Zhu [36] found an $O(n \log n)$ time algorithm with a 1.5 approximation ratio. They use a new data structure, called a *permutation tree* to obtain the $O(n \log n)$ time bound. The beauty and power of a permutation tree is that it arranges the elements of $\pi$ in a heap like tree and can do all that is necessary for a particular transposition in $O(\log n)$ time by doing the necessary "split" and "merge" operations on the tree. Very recently, Firoz *et al.* [40] have shown that the permutation tree data structure also allows the 1.375 approximation algorithm of Elias and Hartman [33] to run in $O(n \log n)$ time.

**Open Problem 33.3** *Now, it would be interesting to see whether the concept of a permutation tree can be used to get an $O(n \log n)$ running time for optimally sorting signed permutations by reversals (see also Open Problem 33.2).*

## 33.5 OTHER OPERATIONS

Once the problems of sorting by reversals and sorting by transpositions received much attention, people started studying variations of these, sometimes motivated by biological applications and somtimes by purely theoretical interests. This section is devoted to some of those variations.

### 33.5.1 Sorting by Prefix Reversals

We start with a variation that in fact was known to the computer scientists before the problems of genome rearrangement surfaced and has additional applications in communication networks. This is the problem of *sorting by prefix reversals*, also popularly known as the *pancake flipping problem.* A *prefix reversal* (also called a *flip*) is a reversal in which the substring to be reversed is a prefix of $\pi$. The problem of *sorting by prefix reversals* [12, 30, 44, 52, 53] deals with finding the minimum number of prefix reversals required to sort a given permutation. This problem was introduced first in 1975 by Dweighter [30], who described the motivation of a chef to rearrange a stack of pancakes from the smallest pancake on the top to the largest one on the bottom by grabbing several pancakes from the top with his spatula and flipping them over, repeating the process as many times as necessary. The first ever attempt to solve this problem was by Gates and Papadimitriou [42]. They proved that the prefix reversal diameter of the symmetric group, $d_{\text{pref}}(n) = \max_{\pi \in S_n} d_{\text{pref}}(\pi)$ is bounded above by $5/3n + 5/3$, where $n$ is the length of $\pi$. They also proved that $n \to \infty \Rightarrow d_{\text{pref}}(n) = 17/16n$.

A well-studied variation of the pancake flipping problem is the *burnt* pancake flipping problem [12, 52, 53] in which each element in the permutation has a sign, and the sign of an element changes with reversals. In communication networks, pancake and burnt pancake networks have a better diameter and a better vertex degree than the popular hypercubes [52]. Some other variations exist of pancake flipping, which provide different efficient interconnection networks [12]. Heydari and Sudborough [51] have claimed that this problem to be NP-complete.

### 33.5.2 Sorting by Prefix Transpositions

A similar variation of sorting by transpositions is *sorting by prefix transpositions*, where a prefix transposition always moves a prefix of the permutation to another location. Dias and Meidanis [28] presented approximation algorithms with ratios 2 and 3 and conjectured that the diameter of prefix transposition of an $n$-element permutation is $n - \lfloor \frac{n}{4} \rfloor$. The complexity of the problem is still unknown.

### 33.5.3 Sorting by Block Interchange

A generalization of transposition is *block interchange*. Remember that a transposition swaps two consecutive blocks (*i.e.*, subsequences). Whereas a *block interchange* can interchange two nonconsecutive subsequences. More formally, given a permutation $\pi = [\pi_0\, \pi_1\, \pi_2\, \ldots \pi_{n+1}]$, a block interchange with parameters $(i, j, k, l)$, where $0 \leq i < j < k < l \leq n + 1$ is applied to $\pi$ by exchanging the blocks $[\pi_i \ldots \pi_{j-1}]$ and $[\pi_k \ldots \pi_{l-1}]$ (the special case of $j = k$, is a transposition) Christie [22] first introduced the problem of *sorting by block interchanges* and showed that unlike sorting by transpositions, this problem can be solved optimally in $O(n^2)$ time.

Recently, Lin *et al.* [65] have studied this sorting problem for circular as well as linear genomes. They have given algorithms that optimally can sort the permutations and have a running time of $O(\delta n)$, where $\delta$ is the minimum number of block interchanges required to sort the permutation and it can be found in $O(n)$ time beforehand. Because $\delta$ can be much less than $n$, their running time is better than the $O(n^2)$ time algorithm of Christie [22]. They also implemented their algorithm and applied it to the circular genomic sequences of three human vibrio pathogens for predicting their evolutionary relationships. The experimental results coincide with the previous ones obtained by others using a different comparative genomics approach, which implies that the block-interchange events seem to play a significant role in the evolution of vibrio species.

**Open Problem 33.4** *It may not be difficult to design an $O(n \log n)$ time algorithm for sorting by block interchanges, but that is still open.*

### 33.5.4 Short Swap and Fixed-Length Reversals

Some other variations of reversals and transpositions exist for sorting genome sequence, and new variations will continue to show up. While analyzing the evolutionary proces of two genome sequnecs, it has been observed that reversals happen among some *fixed-length* subsequences. It motivated scientists to study *sorting by fixed-length reversals*. For example, *soting by short swaps* has been studied by Feng *et al.* [38, 39] and by Heath and Vergara [50], where a *short swap* means a reversal of a subsequence of length two or three. They give several approximation algorithms for this problem. Chen and Skiena [21] studied a more geaneralized version of this problem in which the length of the subsequence is a given integer $k$.

## 33.6 SORTING BY MORE THAN ONE OPERATION

As has been discussed already, the large number of conserved segments in the maps of man and mouse suggest that multiple chromosomal rearrangements have occurred since the divergence of lineages leading to humans and mice. Interestingly enough, in their pioneering paper, Nadeau and Taylor [69] estimated that just 178 ± 39 rearrangements have occurred since this divergence. In spite of a ten-fold increase in the amount of the comparative man/mouse mapping information since then, the new estimate, based on the latest data [25], almost did not change compared with Nadeau and Taylor [69]. Notably, however, for highly rearranged genomes, the scenario may be different. For example, genomes of herpes viruses evolve so rapidly that the similarity between many genes in herpes viruses is very low [57]. In particular, there is little or no cross hybridization between DNAs of the EBV and the HSV-1, and until recently, there was no unambiguous evidence that these herpes viruses actually had a common evolutionary origin [68]. Analysis of such genomes suggest that the evolutions thereof may have involved multiple operations simultaneously.

In this section, we focus on sorting using multiple simultaneous operations. The operations that have been considered for sorting by more than one operations are reversal, transposition, and transreveral. A *transreversal* is a transposition with a reversal applied to a subsequence before it is swapped with another subsequence.

Walter *et al.* [80] provided a two-approximation algorithm for signed permutation for sorting by reversals and transpositions. Gu *et al.* [43] gave a two-approximation algorithm for sorting signed permutations by transpositions and transreversals. Lin and Xue [64] improved this ratio to 1.75 by considering a third operation, called *revrev*, which reverses two contiguous segments. Hartman and Sharan [48] further improved it to 1.5. Blanchette *et al.* [17] worked on a variation of the problem and developed a computer program Derange II built on a greedy algorithm, which attempts to minimize the “weighted sum” of the number of operations. Eriksen [34] provided a $(1 + \epsilon)$ approximation algorithm for sorting signed, circular permutations in which reversals are weighted one and transpositions and transreversals are weighted two.

There has been less progress in the problem of sorting unsigned permutations using more than one rearrangement operations. Walter *et al.* [80] gave a three-approximation algorithm for sorting signed permutations by reversals and transpositions. Rahman *et al.* [71] recently have improved this ratio to $2k$, where $k$ is the approximation ratio of the cycle decomposition algorithm used. Remember that the problem of maximum cycle decomposition of the breakpoint graph $G(\pi)$ of a permutation $\pi$ is NP-hard [19]. Using the best known approximation ratio for this problem, which is by Lin and Jiang [63], their approximation ratio becomes $2.8386 + \epsilon$, for any $\epsilon > 0$.

Interestingly, one may think that the *combined problem* of sorting by more than one operation, say, by reversals and transpositions, equally may be difficult/easy as sorting by one operation (either by reversals or by transpositions), which may lead to a further assumption that approximation algorithms for sorting by a single operation would work fine for sorting by more than one operations. But that is not the case. Because, while sorting a permutation by one operation, at some point, it may be difficult to apply that operation, which can be overcome easily by applying the other operation. On the other hand, the lower bound of the problem of sorting by more than one operation also changes. As a whole, desigining good approximation algorithms for sorting with more than one operation is comparatively difficult. According to Hasan *et al.* [49], a reason for this issue is that a lower bound is determined by the superior operation (*i.e.*, the operation that contributes more toward the sorting process) and that an upper bound of an approximation algorithm in the worst case can be determined by an inferior operation (*i.e.*, the operation that contributes less toward the sorting process), which implies an inferior approximation ratio. They also have derived an improved an adaptive approximation ratios based on the number of inferior operations applied by an optimal algorithm.

### 33.6.1 Unified Operation: Doule Cut and Join

Some other operations exist, such as *fission*, *fusion*, and *translocation* that deal with two sequences. A fission/fussion cuts/inserts one sequence from/into another. A translocation applies two cuts into two sequences and joins the four end points created by these cuts into two different ways as shown in Figure 33.5. Now, observe that these three operations as well as a reversal use two cuts (*i.e.*, a *double cut*) and two joining (*i.e.*, a *double join*). Based on these observations, Yankopulous *et al.* [84] presented a general genome model that includes linear and circular chromosomes and introduced a new operation called *double cut and join* (or shortly DCJ) operation. In addition to inversions and translocations, the DCJ operation also

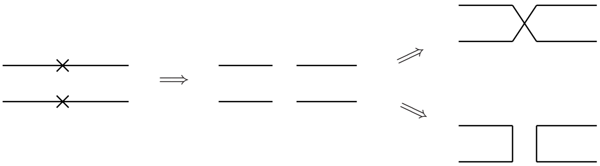


**Figure 33.5** Example of a translocation.

can capture transpositions, transreversals, and block interchanges. By using DCJ, the computation of distance between two permutations becomes easier, and the sorting algorithm becomes efficient [13, 84].

## 33.7 FUTURE RESEARCH DIRECTIONS

In this chapter, we have discussed algorithms for sorting permutations based on different operations. In the corresponding sections, we also have highlighted important open questions that could be considered future avenues for research. We believe these open questions along with many other new issues developing out of newer experimentations would shape the coming days of bioinformatics research.

One interesting avenue for future research, which we did not discuss in this chapter, is the problem of sorting strings instead of sorting permutations. Note carefully that the algorithms and models for genome rearrangements we have discussed so far implicitly assume that each gene is present exactly once in each genome (and, hence, we use permutations). Although this hypothesis of unique genes may be appropriate for small genomes (*e.g.*, viruses), it is clearly unguaranteed for divergent species containing several copies of highly paralogous and orthologous genes scattered across the genomes. Orthologs and paralogs originally were defined by Fitch [41] in 1970. Orthologs are genes in different species that evolved from the same gene in the last common ancestor of the species, and paralogs are genes that were duplicated from a single gene on the same genome. It therefore seems obvious to consider the possibility of having different copies (paralogous) of the same gene in one genome (*e.g.*, multigene families). Several attempts already can be found in the literature to consider genome arrangements problems in which duplicate of genes have been considered (*e.g.*, the exemplar approach [73], evolutionary model involving segment reversals and duplications [31], sorting strings with duplicates [32, 60, 61]; see Table 33.1). We believe that research in this area will continue to flourish in coming years.

Another interesting area to probe into is to consider the more general multibreak rearrangements. Multibreak rearrangements break a genome into multiple fragments and further glue them together in a new order. Although two-break rearrangements represent standard reversals, fusions, fissions, and translocations operations, the three-break rearrangements are a natural generalization of transpositions and inverted transpositions. Multibreak rearrangements in circular genomes were

**Table 33.1 Summary of best known results on genome rearrangement algorithms**

| Operations | Un/signed | $\rho$, NP-hardness | Running Time | Ref. |
|---|---|---|---|---|
| Reversals | Unsigned | NP-hard | — | [19] |
| | | $\rho \not< 1.008$ | — | [16] |
| | | 1.375 | $O(n \log n)$ | [15] |
| | Signed | Optimal | $O(n^{\frac{1}{3}}\sqrt{\log n})$ | [56, 77] |
| | | Optimal | $O(n \log n + kn), k \in O(n)$ | [56] |
| | | Optimal | $O(n^{\frac{1}{3}}\sqrt{\log n})$ | [76] |
| Transpositions | — | 1.375 | $O(n^2)$ | [33] |
| | | 1.5 | $O(n \log n)$ | [36] |
| | | 1.375 | $O(n \log n)$ | [40] |
| Prefix transpositions | — | 2 | — | [28] |
| Block interchange | — | Optimal | $O(n^2)$ | [22, 65] |
| Transpositions, reversals | Signed | 2 | $O(n^2)$ | [80] |
| Transpositions, transreversals | Signed | 1.5 | $O(n^{\frac{3}{2}}\sqrt{\log n})$ | [48] |
| Transpositions, reversals | Unsigned | $2.8386 + \epsilon$ | Polynomial | [71] |

studied in depth in [6] and were applied further to the analysis of chromosomal evolution in mammalian genomes [5]. Later, the study was extended to a linear genome as well [4]. It seems that mutibreak arrangements and the combination of simultaneous rearrangement operations (discussed in Section 33.6) would be one of the main the focus of future research.

## 33.8 NOTES ON SOFTWARE

Although the focus of this chapter was on the theoretical advancement on genome rearrangement algorithms, before we conclude, we would like to discuss very briefly a few related softwares and tools. We note, however, that our discussion in this brief section is in no way complete.

1. Genome Rearrangements In Man and Mouse (GRIMM) is a tool for analyzing rearrangements in pairs of genomes, including unichromosomal and multichromosomal genomes. GRIMM can handle both signed and unsigned data. It began in 2001 as a project by Glenn Tesler and Yang Yu in an undergraduate course taught by Pavel Pevzner's at University of California, San Diego. Subsequent development was done by Glenn Tesler [78, 79].
2. Mauve [27] is a system for efficiently constructing multiple genome alignments in the presence of large-scale evolutionary events such as rearrangement

and inversion. It employs algorithmic techniques that scale well in the amount of sequences being aligned.

3. Genome Inversion and Rearrangement Locator (GRIL) [26] is a tool that can be used to identify the location of rearrangements and inversions in the backbone of a set of DNA sequences. Note, however, that GRIL does not perform an actual alignment of genome sequences.
4. Sorting Permutation by Reversals and block-INterchanGes (SPRING) [1, 66] is a tool to compute the rearrangement distance as well as an optimal scenario between two permutations. SPRING can handle linear/circular chromosomes and can consider reversals, block interchanges, or both. Additionally, it can compute the breakpoint distance between two permutations, which can be used to compare with the rearrangement distance to see whether they are correlated.
5. ROBIN [2, 67] is a web server for analyzing rearrangements between two chromosomal genomes using the block interchange events. It takes two or more linear/circular chromosomes as its input and computes the number of minimum block interchange rearrangements between any two input chromosomes for transforming one chromosome into another and also determines an optimal scenario taking this number of rearrangements.
6. The package baobabLUNA [18] is a Java framework to deal with permutations that represent genomes in rearrangement analysis. It can perform several tasks including building breakpoint graphs, performing reversals, calculating reversal distances, sorting permutations, and so on. Note that, baobabLUNA only can deal with unichromosomal genomes, and its main functionality is the implementation of an algorithm that gives a compact representation of the space of all solutions of the sorting by reversals problem, that is, a representation of all optimal sequences of reversals that sort one genome into another.
7. CTRD [3, 37] is a software for computing translocation distance between genomes. It takes two genomes as its input and tests whether one genome can be transformed into the other. If possible, it computes the translocation distance between two genomes and gives the translocation sequence.

These softwares mostly are available on the internet.

## REFERENCES

1. SPRING: Sorting permutation by reversals and block interchanges. http://algorithm.cs.nthu.edu.tw/tools/SPRING/.
2. ROBIN. A tool for genome rearrangement of block interchanges. http://genome.life.nctu.edu.tw/ROBIN/.
3. CTRD: Computing Signed Translocation Distance. http://www.cs.cityu.edu.hk/∼lwang/software/Translocation/index.html.
4. M.A. Alekseyev. Multi-break rearrangements and breakpoint re-uses: From circular to linear genomes. *J Comput Biol*, 15(8):1117–1131, 2008.

5. M.A. Alekseyev and P.A. Pevzner. Are there rearrangement hotspots in the human genome? *PLoS Comput Biol*, 3(11):e209, 2007.
6. M.A. Alekseyev and P.A. Pevzner. Multi-break rearrangements and chromosomal evolution. *Theor Comput Sci*, 395(2-3):193–202, 2008.
7. D.A. Bader, B.M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *J Comput Biol*, 8(5):483–491, 2001.
8. V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS'93)*, 1993, pp. 148–157.
9. V. Bafna and P. Pevzner. Sorting permutations by transpositions. *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, 1995, pp. 614–623.
10. V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM J Comput*, 25(2):272–289, 1996.
11. V. Bafna and P.A. Pevzner. Sorting by transpositions. *SIAM J Discrete Math*, 11(2):224–240, 1998.
12. D.W. Bass and I.H. Sudborough. Pancake problems with restricted prefix reversals and some corresponding cayley networks. *J Parallel Distr Comput*, 63(3):327–336, 2003.
13. A. Bergeron, J. Mixtacki, and J. Stoye. Hp distance via double cut and join distance. *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM'08)*, volume 5029 of *Lecture Notes in Computer Science*, Springer, New York, 2008, pp. 56–68.
14. P. Berman and S. Hannenhalli. Fast sorting by reversal. In *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM'96)*, volume 1075 of *Lecture Notes in Computer Science*, Sprnger, New York, 1996, pp. 168–185.
15. P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. *Proceedings of the 10th European Symposium on Algorithms (ESA'02)*, volume 2461 of *Lecture Notes in Computer Science*, Springer, New York, 2002, pp. 200–210.
16. P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). *Proceedings of the 26th International Colloquium Automata, Languages and Programming (ICALP'99)*, volume 1644 of *Lecture Notes in Computer Science*, Springer, New York, 1999, pp. 200–209.
17. M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:GC11–17, 1996.
18. M.D.V. Braga. Babobabluna: The solution space of sorting by reversals. *Bioinformatics*, 25(14):1833–1835, 2009.
19. A. Caprara. Sorting by reversals is difficult. *Proceedings of the 1st ACM Conference on Research in Computational Molecular Biology (RECOMB'97)*, 1997, pp. 75–83.
20. A. Caprara and R. Rizzi. Improved approximation for breakpoint graph decomposition and sorting by reversals. *J Combinl Optimi*, 6(2):157–182, 2002.
21. T. Chen and S.S. Skiena. Sorting with fixed-length reversals. *Discrete Appl Math*, 71(1-3): 269–295, 1996.
22. D.A. Christie. Sorting permutations by block-interchanges. *Inform Process Letts*, 60(4):165–169, 1996.

23. D.A. Christie. A 3/2 approximation algorithm for sorting by reversals. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, 1998, pp. 244–252.
24. D.A. Christie. *Genome Rearrangement Problems*. PhD dissertation, University of Glasgow, Glasgow, Scotland, 1999.
25. N.G. Copeland, N.A. Jenkins, D.J. Gilbert, J.T. Eppig, L.J. Maltals, J.C. Miller, W.F. Dietrich, A. Weaver, S.E. Lincoln, R.G. Steen, et al. A genetic linkage map of the mouse: Current applications and future prospects. *Science*, 262:57–65, 1993.
26. A.C. Darling, B. Mau, F.R. Blatter, and N.T. Perna. Gril: Genome rearrangement and inversion locator. *Bioinformatics*, 20(1):122–124, 2004.
27. A.C. Darling, B. Mau, F.R. Blatter, and N.T. Perna. Mauve: Multiple alignment of conserved genomic sequence with rearrangements. *Genome Res*, 14(7):1394–1403, 2004.
28. Z. Dias and J. Meidanis. Sorting by prefix transpositions. *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, volume 2476 of *Lecture Notes in Computer Science*, Springer, New York, 2002, pp. 463–468.
29. T. Dobzhansky and A. Sturtevant. Inversions in the chromosomes of drosophila pseudoobscura. *Genetics*, 23:28–64, 1938.
30. H. Dweighter. Problem E2569. *Am Math Mon*, 82:1010, 1975.
31. N. El-Mabrouk. Reconstructing an ancestral genome using minimum segments duplications and reversals. *J Comput Syst Sci*, 65(3):442–464, 2002.
32. N. El-Mabrouk and D. Sankoff. The reconstruction of doubled genomes. *SIAM J Comput*, 32(3):754–792, 2003.
33. I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *Proceedings of the 5th International Workshop on Algorithms in Bioinformatics (WABI'05)*, volume 3692 of *Lecture Notes in Computer Science*, Springer, New York, 2005, pp. 204–214.
34. N. Eriksen. (1+$\epsilon$)-approximation of sorting by reversals and transpositions. *Theor Comput Sci*, 289(1):517–529, 2002.
35. H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wastlund. Sorting a bridge hand. *Discrete Math*, 241(1-3):289–300, 2001.
36. J. Feng and D. Zhu. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Trans Algorithm*, 3(3):Article 25, 2007.
37. W. Feng, L. Wang, and D. Zhu. Ctrd: A fast applet for computing signed translocation distance between genomes. *Bioinformatics*, 20(17):3256–3257, 2004.
38. X. Feng, Z. Meng, and I.H. Sudborough. Improved upper bound for sorting by short swaps. *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)*, 2004, pp. 98–103.
39. X. Feng, I.H. Sudborough, and E. Lu. A fast algorithm for sorting by short swap. *Proceedings of the Computational and Systems Biology*, 2006, pp. 62–67.
40. J.S. Firoz, M. Hasan, A.Z. Khan, and M.S. Rahman. The 1.375 approximation algorithm for sorting by transpositions can run in $o(n \log n)$ time. *Proceedings of the 4th Annual Workshop on Algorithms and Computation (WALCOM'10)*, 2009. To appear. http://arxiv.org/abs/0910.3292.
41. W.M. Fitch. Distinguishing homologous from analogous proteins. *Syst Zool*, 19:99–113, 1970.

42. W.H. Gates and C.H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Math*, 27:47–57, 1979.

43. Q.P. Gu, S. Peng, and H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theor Comput Sci*, 210(2):327–339, 1999.

44. S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip. *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC'95)*, 1995, 178–189.

45. S. Hannenhalli and R.A. Pevzner. To cut . . . or not to cut (applications of comparative physical maps in molecular evolution). *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*, 1996, pp. 304–313.

46. T. Hartman. A simpler 1.5-approximation algorithm for sorting by transpositions. *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, volume 2676 of *Lecture Notes in Computer Science*, Springer, New York, 2003 pp. 156–169.

47. T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Inform Comput*, 204(2):275–290, 2006.

48. T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals. *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics (WABI'04)*, volume 3240 of *Lecture Notes in Computer Science*, Springer, New York, 2004, pp. 50–61.

49. M. Hasan, A. Rahman, M.S. Rahman, M. Sharmin, and R. Yeasmin. Pancake flipping with two spatulas, 2009. http://arxiv.org/abs/0812.3933v2.

50. L.S. Heath and J.P.C. Vergara. Sorting by short swaps. *J Comput Biol*, 10(5):775–789, 2003.

51. M.H. Heydari and I.H. Sudborough. Sorting by prefix reversals is np-complete. To appear.

52. M.H. Heydari and I.H. Sudborough. On sorting by prefix reversals and the diameter of pancake networks. *Parallel Architectures and Their Efficient Use*, volume 678, 1993, pp. 218–227.

53. M.H. Heydari and I.H. Sudborough. On the diameter of the pancake network. *J Algorithm*, 25(1):67–94, 1997.

54. I. Holyer. The NP-completeness of some edge-partition problems. *SIAM J Comput*, 10(4):713–717, 1981.

55. H. Kaplan, R. Shamir, and R.E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J Comput*, 29:880–892, 1999.

56. H. Kaplan and E. Verbin. Sorting signed permutations by reversals, revisited. *J Comput Syst Sci*, 70(3):321–341, 2005.

57. S. Karlin, E.S. Mocarski, and G.A. Schachtel. Molecular evolution of herpesviruses: Genomic and protein sequence comparisons. *J Virol*, 68:1886–1902, 1994.

58. J. Kececioglu and D. Sankoff. Exact and approximation algorithms for the inversion distance between two permutations. *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM'93)*, volume 684 of *Lecture Notes in Computer Science*, Springer, New York, 1993, pp. 87–105.

59. E.B. Knox, S. Downie, and J.D. Palmer. Chloroplast genome rearrangements and evolution of giant lobelias from herbaceous ancestors. *Mol Biol Evol*, 10:414–430, 1993.

60. P. Kolman and T. Walen. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Appl Math*, 155(3):327–336, 2007.

61. P. Kolman and T. Walen. Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electron J Combinator*, 14(1), 2007.
62. E.V. Koonin and V.V. Dolja. Evolution and taxonomy of positive-strand rna viruses: Implications of comparative analysis of amino acid sequences. *Crit Rev Biochem Molec Biol*, 28:375–430, 1993.
63. G. Lin and T. Jiang. A further improved approximation algorithm for breakpoint graph decomposition. *J Combin Optim*, 8(2):183–194, 2004.
64. G.H. Lin and G. Xue. Signed genome rearrangements by reversals and transpositions: Models and approximations. *Proceedings of the 5th Annual International Conference on Computing and Combinatorics (COCOON'99)*, volume 1627 of *Lecture Notes in Computer Science*, Springer, New York, 1999, pp. 71–80.
65. Y.C. Lin, C.L. Lu, H.-Y. Chang, and C.Y. Tang. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *J Comput Biol*, 12(1):102112, 2005.
66. Y.C. Lin, C.L. Lu, Y.-C. Liu, and C.Y. Tang. Spring: A tool for the analysis of genome rearrangement using reversals and block-interchanges. *Nucleic Acids Res*, 34(Web-Server-Issue):696–699, 2006.
67. C.L. Lu, T.C. Wang, Y.C. Lin, and C.Y. Tang. Robin: a tool for genome rearrangement of block-interchanges. *Bioinformatics*, 21(11):2780–2782, 2005.
68. D.J. McGeoch. Molecular evolution of large dna viruses of eukaryotes. *Semin Virol*, 3:399–408, 1992.
69. J.H. Nadeau and B.A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proc Nat Acad Sci U S A*, 81:814–818, 1984.
70. J.D. Palmer and L.A. Herbon. Tricircular mitochondrial genomes of brassica and raphanus: Reversal of repeat configurations by inversion. *Nucleic Acids Res*, 14:9755–9764, 1986.
71. A. Rahman, S. Shatabda, and M. Hasan. An appoximation algorithm for sorting by reversals and transpositions. *J Discrete Algorithm*, 6(3):449–457, 2008.
72. L.A. Raubenson and R.K. Jansen. Chloroplast dna evidence on the ancient evolutionary split in vrrscular land plants. *Science*, 255:1697–1699, 1992.
73. D. Sankoff. Genome rearrangements with gene families. *Bioinformatics*, 15:909–917, 1999.
74. D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B. Lang, and R. Cedergren. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc Nat Acad Sci U S A*, 89:6575–6579, 1992.
75. A.H. Sturtevant and T. Dobzhansky. Inversions in the third chromosome of wild races of drosophila pseudoobscura, and their use in the study of the history of the species. *Proc Nat Acad Sci U S A*, 22:448–450, 1936.
76. K.M. Swenson, V. Rajan, Y. Lin, and B.M.E. Moret. Sorting signed permutations by inversions in $o(n \log n)$ time. *Proceedings of the 13th Annual International Conference on Research in Computational Molecular Biology (RECOMB'09)*, volume 5541 of *Lecture Notes in Computer Science*, Springer, New York, 2009, pp. 386–399.
77. E. Tannier, A. Bergeron, and M.-F. Sagot. Advances on sorting by reversals. *Discrete App Math*, pp. 881–888, 2007.
78. G. Tesler. Efficient algorithms for multichromosomal genome rearrangements. *J Comput Syst Sci*, 65(3):587–609, 2002.

79. G. Tesler. Grimm: Genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.

80. M.E. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. *Proceedings of the South American Symposium on String Processing and Information Retrieval (SPIRE'98)*, IEEE Computer Society, Washington, DC, 1998, pp. 96–102.

81. M.E.T. Walter, L.R.A.F. Curado, and A.G. Oliveira. Working on the problem of sorting by transpositions on genome rearrangements. *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'03)*, volume 2676 of *Lecture Notes in Computer Science*, Springer, New York, 2003, pp. 372–383.

82. G.A. Watterson, W.J. Ewens, T.E. Hall, and A. Morgan. The chromosome inversion problem. *J Theor Biol*, 99:1–7, 1982.

83. J. Whiting, M. Pliley, J. Farmer, and D. Jeffery. In situ hybridization analysis of chrw mosomal homologies in drosophila melanogaster and drosophila virilis. *Genetics*, 122:90–109, 1989.

84. S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

# 34

# COMPUTING GENOMIC DISTANCES : AN ALGORITHMIC VIEWPOINT

Guillaume Fertin and Irena Rusu

## 34.1 INTRODUCTION

### 34.1.1 What this Chapter is About

Comparative genomics is a field of bioinformatics in which the goal is to compare several species by comparing their genomes, to understand how the different species under study have evolved in time. This study leads, for instance, to reconstructing putative ancestral genomes, building phylogenetic trees, or inferring the functionality of genes or sets of genes.

One of the main activities of comparative genomics consists of comparing *pairs* of genomes to identify their common features and thus also to determine what differentiates them. In that case, genomes usually are modeled as sequences of *genes* in which a gene is identified by a (possibly signed) label. The sign + or –, if present, indicates on which DNA strand the gene lies. In that context, the *order* of the genes in the studied genomes is the main information we are given. Note that the way this order was obtained is out of our scope here; only the order itself is taken into account.

It also should be noted that genomes may contain *several occurrences* of the same gene (possibly carrying different signs if signs are present). In that case, we say that a genome contains *duplicates*. Indeed, genes may be duplicated during evolution, and duplicate genes actually occur frequently in all living species.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Comparing pairs of genomes on that basis can be done roughly in two different ways:

1. Compare the *structure* of the two genomes under study by computing a measure that represents the (dis)similarity between the genomes.
2. Infer the *evolution process* from one genome to another. For this, process, one needs to consider one or several operations (called *rearrangement(s)*) that can occur in a genome during evolution (*e.g.*, inversions or translocations), and the goal is to determine the most parsimonious (*i.e.*, less costly) rearrangement scenario that leads from one genome to the other.

In this chapter, we only focus on option 1. This static viewpoint has the advantage of allowing us to identify conserved regions between genomes, which is not the case with option 2. Note also that, although the term *distance* is used often for option 1. (as is done in the title of this chapter), this only refers to *evolutionary distance* (*i.e.*, the amount of changes that occurred during the evolution process). Indeed, the so-called "distances" that have been defined in the literature are rarely *mathematical distances*, they are *measures* that evaluate the differences and similarities resulting from evolution between the two genomes, either by directly counting the number of changes or, in a complementary way, by counting the conserved regions. Hence, in the following, we often use the term *measure* rather than distance.

The purpose of this chapter is to present some *algorithmic aspects* of pairwise genome comparisons when those comparisons aim at finding a (dis)similarity measure. More precisely, we present several algorithms that were proposed recently for solving (exactly or approximately) several variants of the problem. Our goal is not to survey exhaustively all existing results on that topic but rather to give a sample of different algorithmic ideas and techniques that have been used to answer some problems. Besides the fact that it presents original and nontrivial concepts that we think are of interest for the reader, it also gives a flavor of the inventiveness and the richness of recent research on the subject.

### 34.1.2 Definitions and Notations

Genomes under consideration in this chapter are represented as sequences of (possibly signed) integers built from the alphabet $\Sigma = \{1, 2, 3 \ldots, n\}$, where $n$ is as large as necessary. When *unsigned* (respectively, *signed*) genomes are considered, then their representation is a sequence of unsigned (respectively, signed) integers. When a sequence contains distinct integers (*i.e.*, the corresponding genome has no duplicates), the sequence is called a *permutation*, whereas in the contrary case, it is called a *string*. For instance, $P = (2\ -3\,8\ -4\ -5\,1\,7\ -6)$ is a signed permutation, whereas $Q = (3\ -4\ -3\,2\,1\,2\,2)$ is a signed string.

For any genome $P$, its length (*i.e.*, its number of genes) is denoted as $m_P$. Moreover, for any $1 \le i \le j \le m_P$, $P[i]$ denotes the $i$th element of $P$, $|P[i]|$ is $P[i]$ whose sign has been removed, and $P[i, j] = (P[i]\,P[i+1]\,\ldots\,P[j])$ denotes the portion of $P$ whose extremities are given by indices $i$ and $j$, both being included. For instance, if $P = (2\ -3\ 8\ -4\ -5\ 1\ 7\ -6)$, then $P[2] = -3$, $|P[2]| = 3$, and $P[2, 4] = (-3\ 8\ -4)$.

A *duo* in a genome $P$ is a set of two consecutive elements of $P$. For any $1 \leq i \leq m_P - 1$, the duo $d_i$ represents $P[i]P[i + 1]$ and is denoted as follows: $d_i = (P[i], P[i + 1])$. Two duos $d = (a, b)$ and $d' = (a', b')$ of a signed genome are said to be *identical* if (i) $a = a'$ and $b = b'$ or (ii) $a = -b'$ and $b = -a'$. If the genome is unsigned, then two such duos are identical whenever $a = a'$ and $b = b'$ only. Given two permutations $P$ and $Q$ built on the same alphabet, an *adjacency* in $P$ is a duo $d$ for which a duo $d'$ exists in $Q$ such that $d$ and $d'$ are identical. Whenever $d$ is not an adjacency in $P$, then it is a *breakpoint*. We note that these two notions are symmetric; that is, given two permutations $P$ and $Q$ built on the same alphabet, the number of adjacencies (respectively, breakpoints) in $P$ is equal to the number of adjacencies (respectively, breakpoints) in $Q$.

### 34.1.3 Organization of the Chapter

This chapter is organized as follows: in Section 34.2, we are interested in comparing pairs of genomes by finding their common or conserved intervals. In this context, three algorithms are presented. Section 34.3 is devoted to two algorithms to determine the minimum number of breakpoints between pairs of genomes containing duplicates. Section 34.4 is the conclusion.

## 34.2 INTERVAL-BASED CRITERIA

### 34.2.1 Brief Introduction

Breakpoints and adjacencies are easy to compute in permutations but do not give much insight on the genome organization in terms of sets of genes that are close to each other in both genomes. However, these groups of genes are particularly interesting because they show a similarity between genomes in their content, a similarity that has been preserved despite of the past evolutionary events.

Looking for *intervals* rather than *duos* allows us to model such regions with identical content but different gene order. In this context, the location of the genes changes from one genome to the other; some genes even may get duplicated, but the new locations and the possible duplicates still form an interval in the second genome.

Going further, that is, speaking about *approximate intervals* (substrings with almost identical content) is possible, but measuring the similarity or dissimilarity between genomes with this approach becomes difficult. Each of these regions is seen more successfully like a *cluster of genes* sharing a potential common function. Their importance is undeniable, but it is not the point of this chapter. This is why we limit our study to measures based on intervals.

### 34.2.2 The Context and the Problems

Let $P$ be a signed string of integers over the finite alphabet $\Sigma$, representing a linear genome.

**Definitio 34.1** [12] *The **character set** of the interval $P[i, j]$ of $P$, with $1 \leq i \leq j \leq m_P$, is defined as*

$$\mathcal{CS}(P[i, j]) = \{|P[h]| \,:\, i \leq h \leq j\}$$

The character set of an interval stores the content of the interval, regardless of the order of genes within it, their signs, or their number of occurrences. When the same character set is defined by two intervals of two strings, a strong local similarity is identified between the two strings.

**Definitio 34.2** [12] *Let $\mathcal{C} \subseteq \Sigma$ be a set of integers and $P$, $Q$ be two signed strings over $\Sigma$. Set $\mathcal{C}$ is a **common interval** of $P$ and $Q$ if positions $a, b, i, j$ exist, with $1 \leq a \leq b \leq m_P$ and $1 \leq i \leq j \leq m_Q$, such that*

$$\mathcal{CS}(P[a, b]) = \mathcal{CS}(Q[i, j]) = \mathcal{C}$$

This definition allows several variants:

1. Common intervals of two permutations, defined in [14], when $P$ and $Q$ are permutations on $\{1, 2, \ldots, n\}$
2. Conserved intervals of two permutations, defined in [4], when $P$ and $Q$ are signed permutations on $\{1, 2, \ldots, n\}$ and the common interval is required to satisfy either $P[a] = Q[i]$ and $P[b] = Q[j]$, or $P[a] = -Q[j]$ and $P[b] = -Q[i]$
3. Common intervals of two strings, defined in [12], when $P$ and $Q$ are unsigned strings with elements in $\{1, 2, \ldots, n\}$.

Each variant defines a criterion to measure the similarity between strings $P$ and $Q$ as the number of common/conserved intervals of $P$ and $Q$. For common intervals and conserved intervals in permutations, a mathematically rigorous notion of distance (satisfying the three properties of a metric) may be defined as follows. Note that the number of common intervals between a permutation $P$ and itself is the number of intervals of $P$, that is, $n(n + 1)/2$. The same holds for conserved intervals.

**Definitio 34.3** *Let **Common(P, Q), Conserved(P, Q)** be the number of common and conserved intervals of two permutations $P$ and $Q$, respectively. The **common intervals distance** between $P$ and $Q$ is defined as follows:*

$$distance_{Common}(P, Q) = n(n + 1) - 2\,Common(P, Q)$$

*The **conserved intervals distance** between P and Q (assumed signed) is defined as follows*:

$$\text{distance}_{\text{Conserved}}(P, Q) = n(n+1) - 2\,\text{Conserved}(P, Q)$$

This notion of distance was introduced in [4] for conserved intervals, and in the more general case where each of $P$ and $Q$ is replaced by a set of permutations.

The extension of the definitions and methods used in this section to more than two strings is briefly discussed in Section 34.2.6. For common intervals in strings, the possible difference between the lengths of $P$ and $Q$ as well as the possibly different number of locations of each interval in each string make a neat definition much more difficult to find.

In the next three sections, we present algorithms to compute the number of common intervals between $P$ and $Q$ according to each of these three variants. These three algorithms allow us to see and discuss three different ways to reach a similar goal. The reader will note that for common intervals, both in permutations and in strings, the algorithms given here really compute each of the searched intervals. The number of intervals then is computed implicitly. For conserved intervals, it is possible to compute directly the number of intervals without displaying them all.

To understand the differences between the three approaches, let us start with a slightly deeper analysis of the problem. Looking for a common interval of $P$ and $Q$ is looking for two positions $i$ and $j$ on $Q$ such that the elements in the interval $Q[i, j]$ also form an interval in $P$.

**Definitio 34.4** *Given a position $i$ in $Q$, a **Max zone** of $i$ is any maximal interval of $P$ whose character set contains $Q[i]$ and is included in $\mathcal{CS}(Q[i, m_Q])$. A **Min zone** of $i$ is any maximal interval of $P$ whose character set contains $Q[i]$ and is included in $\mathcal{CS}(Q[1, i])$.*

Recalling that the elements in the interval $Q[i, j]$ are both in $\mathcal{CS}(Q[i, m_Q])$ and in $\mathcal{CS}(Q[1, j])$, the following lemma is then easy to deduce:

**Lemma 34.1** *The set $\mathcal{C}$ defined by $\mathcal{C} = \mathcal{CS}(Q[i, j])$, with $1 \le i \le j \le m_Q$, is a common interval of $P$ and $Q$ if and only if a Max zone of $i$ exists and a Min zone of $j$ exists whose intersection has character set $\mathcal{C}$.*

Figure 34.1 illustrates this lemma. Note that when $P$ and $Q$ are permutations (with or without signs), there is exactly one Max zone and one Min zone for every position in $Q$. The three approaches are summarized as follows.

*Commuting Generators.* The algorithm for common intervals in permutations, presented in Section 34.2.3 and introduced in [3], identifies for every position $i$ in $Q$ (renumbered such that $Q$ is the identity permutation), a subzone of the Max zone and a subzone of the Min zone containing consecutive elements (including $i$) with respect to the order in $Q$. These subzones are called *generators*.

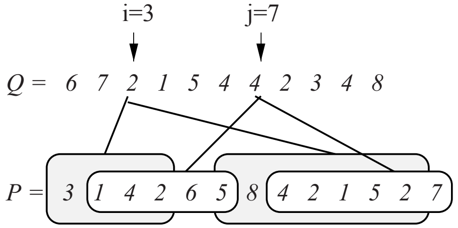


**Figure 34.1** Max zones (grey boxes) of position 3 in $Q$ and Min zones (white boxes) of position 7 in $Q$, when $Q = (6\,7\,2\,1\,5\,4\,4\,2\,3\,4\,8)$ and $P = (3\,1\,4\,2\,6\,5\,8\,4\,2\,1\,5\,2\,7)$. Sets $\{1, 2, 4\}$, $\{5\}$, and $\{1, 2, 4, 5\}$ all may be obtained as the intersection of a Max zone of 3 with a Min zone of 7, but only $\{1, 2, 4, 5\}$ is $\mathcal{CS}(Q[3, 7])$ and is thus a common interval of $P$ and $Q$.

A common interval then is defined by any pair $i, j$ such that $j$ is in the Max subzone of $i$, and $i$ is in the Min subzone of $j$. Once the subzones are computed, obtaining all common intervals is an easy task. The resulting algorithm is in $O(n + N)$, where $n = m_P = m_Q = \text{Card}(\Sigma)$ is the common length of $P$ and $Q$, and $N$ is the number of common intervals between $P$ and $Q$.

*Bound-and-Drop*. The algorithm for conserved intervals in (signed) permutations, presented in Section 34.2.4 and introduced in [4], considers for each $j$ in $Q$ (renumbered such that $Q$ is the identity permutation) the candidates $i < j$ in $Q$ such that $i$ is in the Min zone of $j$, $j$ is in the Max zone of $i$, and the number of elements in the interval of $P$ with endpoints $i$ and $j$ is $j - i + 1$. The bad candidates $i$ are dropped, whereas the first suitable one found is validated. The result is a set of special conserved intervals $\mathcal{CS}([i, j])$, called *irreducible intervals*, which allows quickly computing, in $O(n)$, the total number of conserved intervals. An algorithm to find all conserved intervals in $O(n + N)$ time then is obtained easily (where $n = m_P = m_Q = Card(\Sigma)$).

*Element Plotting*. The algorithm for common intervals in strings, presented in Section 34.2.5 and introduced in [12], deals with multiple Max zones and Min zones for any fixed $i$ by considering in a left-to-right order all positions $j > i$ and plotting on $P$ the elements found in $Q[i, j]$. The Max zones of $i$ and the Min zones of $j$ are computed in this way simultaneously but uncontiguously and incompletely (only the useful elements, those in $Q[i, j]$, are plotted). When an interval of plotted elements in $P$ has the same number of distinct elements as $\mathcal{CS}(Q[i, j])$, a common interval is displayed. This is equivalent to saying that the intersection of a Max zone of $i$ and a Min zone of $j$ is an interval that contains exactly the elements in $\mathcal{CS}(Q[i, j])$. The algorithm presented here runs in $\Theta(m^2)$ (where $m = \max\{m_P, m_Q\}$) but may be improved to $O(m^2)$ as shown in [12].

### 34.2.3 Common Intervals in Permutations and the Commuting Generators Strategy

Common intervals of two permutations were introduced in [14], together with a first (and complex) algorithm in $O(n + N)$ to compute them.

The algorithm we present in this section was proposed in [3]. The genomes $P$ and $Q$ are represented as permutations on $\Sigma = \{1, 2, \ldots, n\}$. Moreover, to simplify the presentation, we assume without loss of generality that $Q$ is the identity permutation, denoted $Id$, and $P$ is an arbitrary permutation. This can be achieved easily given two arbitrary permutations by renumbering $Q$ to obtain $Id$ and renumbering $P$ accordingly. As a consequence, given $i \in \{1, 2, \ldots, n\}$, we have that $Q[i] = Id[i] = i$ and that the Max zone and Min zone of $i$ may be redefined as follows:

**Definitio 34.5** *Given $i \in \{1, 2, \ldots, n\}$, define on $P$ the following intervals:*

$I\text{Max}[i]$: the largest interval containing $i$ and elements greater than $i$
$I\text{Min}[i]$: the largest interval containing $i$ and elements smaller that $i$

Let $(i..j)$ be a shorter notation for $\mathcal{CS}(Id[i, j])$ that assumes $1 \leq i \leq j \leq n$. According to Lemma 34.1, a set $(i..j)$ is a common interval of $P$ and $Id$ if and only if the equality $\mathcal{CS}(I\text{Max}[i]) \cap \mathcal{CS}(I\text{Min}[j]) = (i..j)$ holds. With the supplementary notation

Sup$[i]$: the largest integer such that $(i..\text{Sup}[i]) \subseteq \mathcal{CS}(I\text{Max}[i])$, and
Inf$[i]$: the smallest integer such that $(\text{Inf}[i]..i) \subseteq \mathcal{CS}(I\text{Min}[i])$

the latter equality holds if and only if $j \leq \text{Sup}[i]$ and $\text{Inf}[j] \leq i$. Equivalently, we have that $(i..j)$ is a common interval of $P$ and $Id$ if and only if

$$(i..j) = (i..\text{Sup}[i]) \cap (\text{Inf}[j]..j)$$

The vectors Sup and Inf, both of size $n$, are thus sufficient to generate all common intervals $(i..j)$ (and only the common intervals) using this formula. But this pair of vectors is not necessarily unique.

A general definition may be given:

**Definitio 34.6** *A pair $(R, L)$ of vectors of size $n$ is a* ***generator*** *for the common intervals of $P$ and $Id$ if the following properties hold:*

1. *$R[i] \geq i$ and $L[i] \leq i$, for all $1 \leq i \leq n$, and*
2. *$(i..j)$ is a common interval of $P$ and $Id$ if and only if*

$$(i..j) = (i..R[i]) \cap (L[j]..j)$$

The pair (Sup, Inf) allows us to answer affirmatively the question whether a generator exists for every $P$ and $Id$. Paper [3] deeply analyzes generators for two or more permutations, and for other families of intervals. We focus here on computing the common intervals of two permutations, which is done with Algorithm 34.1.

**Algorithm 34.1** Algorithm Common_Intervals_In_Permutations [3]

```
Step 1. For all $i \in \{1, 2, \ldots, n\}$ do compute $I\mathrm{Max}[i]$ and $I\mathrm{Min}[i]$ Endfor;
Step 2. Compute $(\mathrm{Sup}, \mathrm{Inf})$;
Step 3. Compute common intervals.
```

Consider Steps 1 through 3 one by one.

STEP 1

As $I\mathrm{Max}[i]$ and $I\mathrm{Min}[i]$ are intervals of $P$, computing them just requires computing their left and right endpoints. To ensure a linear complexity, all endpoints of a given type are obtained during a single search along $P$. To obtain, for instance, the left endpoints $L\mathrm{Min}[i]$ of $I\mathrm{Min}[i]$ $(1 \leq i \leq n)$, Algorithm 34.2 is proposed, which uses a stack $S$ to store in a convenient order the current candidates. The linear running time of this algorithm is obvious.

**Algorithm 34.2** Algorithm Step 1 (LMin version) [3]

```
{$S$ is a (initially empty) stack of positions in $P$}
Stack 0 on S; $P[0] \leftarrow n+1$;
For $h = 1$ to $n$ do
    While $P[\mathrm{top}(S)] < P[h]$ do Unstack $\mathrm{top}(S)$ Endwhile;
    $L\mathrm{Min}[P[h]] \leftarrow \mathrm{top}(S) + 1$;
    Stack $h$ on $S$;
Endfor.
```

STEP 2

Computing $(\mathrm{Sup}, \mathrm{Inf})$ when the endpoints of $I\mathrm{Max}[i]$ and $I\mathrm{Min}[i]$ are known is based on the following property:

**Algorithm 34.3** Algorithm Step 2 [3]

```
{$W, w$ are two vectors of size $n$}
$\mathrm{Inf}[1] \leftarrow 1$; $\mathrm{Sup}[n] \leftarrow n$;
For $i = 1$ to $n$ do $W[i] \leftarrow i$; $w[i] \leftarrow i$ Endfor;
For $i = n-1$ downto 1 do
    While $W[i]+1 \in \mathcal{CS}(I\mathrm{Max}[i])$ do $W[i] \leftarrow W[W[i]+1]$ Endwhile;
    $\mathrm{Sup}[i] \leftarrow W[i]$
Endfor;
For $i = 2$ to $n$ do
    While $w[i]-1 \in \mathcal{CS}(I\mathrm{Min}[i])$ do $w[i] \leftarrow w[w[i]-1]$ Endwhile;
    $\mathrm{Inf}[i] \leftarrow w[i]$
Endfor.
```

**Lemma 34.2** *Let $P$ be a permutation. If $(i..k) \subseteq \mathcal{CS}(IMax[i])$, then $Sup[i] \geq Sup[k]$. If $(k..i) \subseteq \mathcal{CS}(I\mathrm{Min}[i])$, then $Inf[i] \leq Inf[k]$.*

The proof of this lemma is done easily by noting that we have $(k..\mathrm{Sup}[k]) \subseteq \mathcal{CS}(I\mathrm{Max}[k])$ and that $I\mathrm{Max}[k]$ is included in $I\mathrm{Max}[i]$. Then both $(i..k)$ and $(k..\mathrm{Sup}[k])$ are subsets of $\mathcal{CS}(I\mathrm{Max}[i])$ and so is their union. The conclusion follows from the definition of $\mathrm{Sup}[i]$. A similar reasoning is valid for the second part of the lemma.

Lemma 34.2 allows us to compute Sup$[i]$ in the decreasing order of $i$. Assuming Sup$[i+1], \ldots,$ Sup$[n]$ are known already, one obtains Sup$[i]$ by initializing it with $i$ and successively updating it to Sup$[k_h]$ as long as the element $k_h$, defined by $k_1 = i+1$, $k_{h+1} =$ Sup$[k_h] + 1$ $(h \geq 1)$, is found in $I$Max$[i]$. Algorithm 34.3 is based on this idea and runs in $O(n)$ because the total number of updates over all $i$ is $n-1$.

STEP 3
To compute common intervals, a *commuting* generator is needed to ensure a minimum running time for the algorithm.

**Definitio 34.7** *A generator* $(R, L)$ *is* ***commuting*** *if each of the collections of sets* $\{(i..R[i]) : 1 \leq i \leq n\}$ *and* $\{(L[i]..i) : 1 \leq i \leq n\}$ *has the property that any two distinct sets of the collection are either disjoint, or one of them contains the other one.*

The generator (Sup, Inf) computed in Step 2 is commuting (easily deduced either using the definition or Algorithm 34.3), but Algorithm 34.4 works as well for an arbitrary commuting generator. It first computes for every element (and position) $i > 1$ in $Id$ a value Support$[i]$, which gives the rightmost position $h < i$ such that $R[h] \geq R[i]$. Then, for each $j$ in decreasing order, it successively identifies the positions $i =$ Support$^q[j]$ $(q \geq 0$, with the convention that Support$^0[j] = j$, and Support$^r[j] =$ Support[Support$^{r-1}[j]$]) such that $L[j] \leq i$, that is, $(L[j]..j)$ contains $i$. This is sufficient to ensure that $(i..j)$ is a common interval because we also have, by the definition of the vector Support, that $R[i] \geq j$, and thus, $(i..j) = (i..R[i]) \cap (L[j]..j)$.

**Algorithm 34.4** Algorithm Step 3 [3]

```
{$S$ is a (initially empty) stack of positions in $Id$}
{$R, L$ are two vectors of size $n$, representing the given generator}
Stack 1 on $S$
For $i = 2$ to $n$ do
   While $R[top(S)] < i$ do Unstack $top(S)$ Endwhile;
   Support$[i] \leftarrow top[S]$;
   Stack $i$ on $S$
Endfor;
For $j = n$ downto 1 do
   $i \leftarrow j$;
   While $i \geq L[j]$ do
     Output the common interval $(i..j)$;
     $i \leftarrow$ Support$[i]$
   Endwhile
Endfor.
```

The second *while* loop of this algorithm is executed proportionally to the number of common intervals it outputs so that the running time of the algorithm is in $O(n + N)$.

The algorithm COMMON_INTERVALS_IN_PERMUTATIONS we described in this section first collects the necessary information and then builds at once all common intervals in contrast to the next algorithms, which use sequentially collected information to display sequentially the common intervals.

### 34.2.4 Conserved Intervals in Permutations and the Bound-and-Drop Strategy

In [4], conserved intervals were introduced as a family of common intervals that should not be broken by rearrangement operations on the genome. An $O(n)$ algorithm to compute the number of conserved intervals between two (and even more, see Section 34.2.6) permutations is given in the paper, and we present it subsequently. Displaying all conserved intervals in $O(n + N)$, where $N$ is the total number of conserved intervals, is an easy task using Lemma 34.3 and the algorithm.

In this section, genomes are *signed* permutations on $\{1, 2, \ldots, n\}$. Without loss of generality, we assume once again that one of the permutations is the identity permutation $Id$, and the other permutation $P$ is an arbitrary *signed* permutation. Moreover because singletons are known conserved intervals, they are omitted from the presentation. All conserved intervals considered in the remaining of this section are therefore, by definition, nonsingletons.

**Definitio 34.8** *Let $P$ be a signed permutation. A conserved interval $C$ of $Id$ and $P$ is* ***reducible*** *if smaller conserved intervals $C_1, C_2, \ldots, C_h$ ($h \geq 2$) exist such that $C$ is the union of $C_1, C_2, \ldots, C_h$. In the contrary case, $C$ is called* ***irreducible****.*

Note that irreducible conserved intervals are not necessarily minimal with respect to inclusion. Moreover, it is easy to prove that irreducible conserved intervals are either disjoint, are included in each other (and with different endpoints), or are overlapping on exactly one element so that they form chains:

**Definitio 34.9** *Let $P$ be a signed permutation. A collection $C_1, C_2, \ldots, C_l$ ($l \geq 1$) of irreducible conserved intervals of $P$ and $Id$ is a* ***chain*** *if $C_x$ and $C_{x+1}$ have exactly one element in common, for all $x$, $1 \leq x \leq l - 1$. A chain $C_1, C_2, \ldots, C_l$ ($l \geq 1$) is* ***maximal*** *if no irreducible conserved interval $C_0$ exists such that $C_0, C_1, C_2, \ldots, C_l$ or $C_1, C_2, \ldots, C_l, C_0$ is a chain.*

Maximal chains partition the collection of irreducible conserved intervals. Moreover, a conserved interval is always a chain (not necessarily maximal) so that estimating the number of conserved intervals and displaying each of them when irreducible conserved intervals are known may be done using the following result:

**Lemma 34.3** *Let $P$ be a signed permutation. A maximal chain $C_1, C_2, \ldots, C_l$ ($l \geq 1$) of irreducible conserved intervals of $P$ and $Id$ generates exactly $l(l + 1)/2$ conserved intervals.*

It remains to give the algorithm for finding the irreducible conserved intervals. Note that the conserved intervals with endpoints $i, j$, $1 \leq i \leq j \leq n$, on $Id$ have either positive endpoints $i, j$ (in this order, from left to right) or negative endpoints

$-j$, $-i$ (in this order, from left to right) on $P$. Algorithm 34.5 shows how to identify the irreducible conserved intervals with positive endpoints (called *positive irreducible intervals*). The same algorithm, applied to $Id$ and to the result $\overline{P}$ of a complete signed reversal on $P$ (*i.e.*, $\overline{P} = (-P[n]\ -P[n-1]\ \ldots -P[1])$) identifies the irreducible conserved intervals with negative endpoints (called *negative irreducible intervals*).

**Algorithm 34.5** Algorithm Positive_Irreducible_Intervals [4]

```
{$S$ is a (initially empty) stack of indices in $P^*$}
{$B$ is a vector of size $n+2$}
Stack 0 on $S$; $B[0] \leftarrow n+1$;
Compute $L\text{Min}[i]$, $i = 0, \ldots, n+1$, using Algorithm Step 1 (LMin version);
For $j = 1$ to $n+1$ do
   If $L\text{Min}[|P^*[j]|] > 1$ then $B[j] \leftarrow |P^*[L\text{Min}[j]-1]|$ else $B[j] \leftarrow n+1$ Endif;
   While $|P^*[j]| < P^*[\text{top}(S)]$ or $|P^*[j]| > B[\text{top}(S)]$ do
     Unstack $top(S)$
   Endwhile;
   If $j - \text{top}(S) = P^*[j] - P^*[\text{top}(S)]$ and $B[j] = B[\text{top}(S)]$ then
      Output the positive irreducible interval $(P^*[\text{top}(S)]..P^*[j])$
   Endif;
   If $P^*[j] > 0$ then Stack $j$ on $S$ Endif;
Endfor.
```

For simplicity reasons, the algorithm works on the permutation $P^* = (0\ P[1]\ \ldots P[n]\ n+1)$. The interpretation of the results on the initial permutation $P$ is easy and left to the reader. Moreover, $L\text{Min}[i]$ is the left endpoint of the interval $I\text{Min}[i]$, defined as in Definition 34.5, but for $P^*_+$, the unsigned permutation that is obtained from $P^*$ by removing the signs.

It is important to note that each index $j$ in $P$ can be the right endpoint of at most one positive irreducible interval. Then, each $j$ is considered, in increasing order, and the corresponding left endpoint of the possible interval is searched for on the stack $S$, which contains the positions of the candidates (both the positions and their corresponding elements in $P^*$ are in increasing order from bottom to top). To this end, a value $B[i]$ is computed for each $i$ that upperly bounds the values $|P^*[j]|$ obtained at a possible right end $j$ of a conserved interval $(i..j)$. Obviously bad candidates top($S$) (too large or whose bound $B[\text{top}(S)]$ is exceeded by $|P^*[j]|$) are dropped, and the next candidate either is the suitable one (the number of elements in the interval is correct, and these elements are all smaller than $P^*[j]$), or is not, and in this latter case, there is no suitable candidate.

The running time of this algorithm is in $O(n)$, because the While loop will unstack globally at most $n$ elements (each index is stacked exactly once on $S$).

### 34.2.5 Common Intervals in Strings and the Element Plotting Strategy

In this section, $P$ and $Q$ are unsigned strings over $\{1, 2, \ldots, n\}$, of respective lengths $m_P$ and $m_Q$, which implies that every element in the alphabet may have zero, one, or several occurrences in each string. Without loss of generality, it is assumed that

$n \leq m_P + m_Q$ (otherwise, a renumbering of the elements in the alphabet may be performed to achieve this), and that strings $P$ and $Q$ are extended at their left and right extremities with a new element (not in $\Sigma$), say $n + 1$. To simplify explanations, the resulting strings still are noted $P$ and $Q$ with lengths $m_P$ and $m_Q$ (which only differ by two from the initial lengths, thus not affecting the complexity order of Algorithm 34.6).

The algorithm presented in this section was proposed in [12] and uses a very different strategy to display all common intervals compared with the ones in Sections 34.2.3 and 34.2.4. To start with, note that we may limit the searches to *maximal locations* of common intervals:

**Definitio 34.10** *Let $Q$ be an unsigned string on the alphabet $\Sigma = \{1, 2, \ldots, n\}$ and $\mathcal{C} \subseteq \{1, 2, \ldots, n\}$. Interval $Q[i, j]$ is a* ***location*** *of $\mathcal{C}$ in $S$ if $\mathcal{CS}(Q[i, j]) = \mathcal{C}$. The location $Q[i, j]$ is* ***left maximal*** *if $i = 1$ or $Q[i-1] \notin \mathcal{C}$, is* ***right maximal*** *if $j = m_Q$ or $Q[j+1] \notin \mathcal{C}$, and is* ***maximal*** *if it is both left maximal and right maximal.*

Algorithm 34.6 uses a vector of lists POS and a matrix NUM to store, respectively, the positions in $P$ of every element $c \in \{1, 2, \ldots, n\}$ and the number of distinct elements in each interval $P[a, b]$ with $a, b \in \mathcal{CS}(P)$. For each pair of indices $i, j$ with $1 \leq i \leq j \leq m_Q$, such that $Q[i, j]$ is a maximal location of $\mathcal{CS}(Q[i, j])$ (set stored as a vector OCC in the algorithm), the $n_{\mathrm{OCC}}$ elements of $\mathcal{CS}(Q[i, j])$ are plotted (or marked) on $P$. Intervals of plotted elements on $P$ then are tested to see whether they have all the desired elements (*i.e.*, $n_{\mathrm{OCC}}$ elements) and whether they are maximal. In the affirmative case, they are maximal common intervals and are thus output by the algorithm.

**Algorithm 34.6** Algorithm Common_Intervals_In_Strings [12]

```
{$\mathrm{OCC}[c] = 1$ if and only if $c$ belongs to the current interval $Q[i, j]$}

Compute data structures POS and NUM for $P$;
For $i = 1$ to $m_Q$ do
   For $c = 1$ to $n$ do $\mathrm{OCC}[c] \leftarrow 0$ Endfor;
   $n_{\mathrm{OCC}} \leftarrow 0$; $j \leftarrow i$;
   While $j \leq m_Q$ and $Q[i, j]$ is left maximal do
     $c \leftarrow Q[j]$;
     $\mathrm{OCC}[c] \leftarrow 1$; $n_{\mathrm{OCC}} \leftarrow n_{\mathrm{OCC}} + 1$;
     While $Q[i, j]$ is not right maximal do $j \leftarrow j + 1$ Endwhile;
     For all $p$ in $\mathrm{POS}[c]$ do
       Mark element $c$ at position $p$ in $P$;
       $P[a, b] \leftarrow$ the largest interval of marked characters with $a \leq p \leq b$;
       If $\mathrm{NUM}[a, b] = n_{\mathrm{OCC}}$ and $P[a, b]$ is maximal then
         Output $\mathcal{C} = \mathcal{CS}(Q[i, j])$ and the pair $(P[a, b], Q[i, j])$
       Endif
     Endfor;
     $j \leftarrow j + 1$
   Endwhile
EndFor.
```

It is easy to imagine examples on which this algorithm will display twice (or more) the same common interval $\mathcal{C}$ with two different locations either on $P$ or on $Q$. The algorithm may be modified to avoid redundant outputs, as shown in [12].

This algorithm runs in $\Theta(m^2)$, where $m = \max\{m_P, m_Q\}$, but a variant of it exists that runs in $O(m^2)$ [12].

### 34.2.6 Variants

The notions and algorithms presented so far either are devised directly for or extended easily to an arbitrary number $K \geq 2$ of genomes (see [3, 4, 12]). In this case, the complexity becomes $O(Kn + N)$ to output all common or all conserved intervals in (signed) permutations $O(Kn)$ to compute the number of conserved intervals in signed permutations, and $O(Km^2)$ to output all common intervals in strings.

The case of genomes with duplicates, represented by signed or unsigned strings, was approached in Section 34.2.5 under the double hypothesis that (i) no distinction can be made among duplicates in $P$ and in $Q$ and that (ii) the locations of a common interval of $P$ and $Q$ may contain an arbitrary number of copies of each gene. This approach passes up the underlining biological hypothesis that the copies of a gene are obtained during speciation and duplication processes that imply relationships between duplicates. Several ways to express this biological hypothesis exist (see [6] for detailed explanations), resulting in different hard problems to solve, for which different approaches were proposed.

In our next section, we present some of them.

## 34.3 CHARACTER-BASED CRITERIA

### 34.3.1 Introduction and Definition of the Problems

As mentioned in the previous section, computing the number of breakpoints between two genomes that do not contain duplicates is an easy task. On the contrary, when duplicates occur in genomes, an intuitive way of dealing with them is to get back to permutations, that is, genomes without duplicates. For this, the goal is to establish a one-to-one correspondence between genes of both genomes, that is a *matching*, say $\mathcal{M}$. Once $\mathcal{M}$ is found, we remove from both genomes the genes that are not matched by $\mathcal{M}$ (this happens, for instance, when the number of duplicates of a given gene differs between both genomes), and after a renaming of the genes, we obtain a permutation on which all classical measures can be computed.

The tricky part of the process is to find an appropriate matching. Usually, the matching $\mathcal{M}$ that we look for is one that optimizes the studied measure, thus following the parsimony hypothesis, which states that nature always chooses the "shortest path" to go from one species (*i.e.*, one genome) to another.

In that case, the problem of computing a measure between two genomes, which was just a computation problem in permutations (we just are asked to provide a number) becomes an *optimization* problem in strings in which one wants to find the matching that optimizes the studied measure.

In the following, we are interested in genomes that contain duplicates. We first look at the particular case in which the pairs of genomes that we compare contain, for each gene $g$, exactly the same number of copies of $g$. In that case, we say that genomes are *balanced*. This restriction could seem strong, but genes are DNA fragments, and no two genes are constituted of the exact same DNA sequence; thus, duplicate genes are actually genes that are pairwise *similar* (*i.e.*, their DNA sequences are sufficiently close). Hence, it is always possible to build gene clusters such that there are as many copies in the first genome as in the second (*e.g.*, by removing from a given cluster those genes that are less similar to the others).

Suppose that the two input genomes are balanced. It thus seems natural to ask for a one-to-one correspondence of the genes (*i.e.*, a matching) that contains *all* the genes of both genomes. Such a matching is referred to in the following as a *full matching*.

Now, if the two input genomes are *unbalanced*, then we need to define more precisely the matching that we look for. First, for any gene $g$, we denote by $\mathsf{occ}(g,P)$ (respectively, $\mathsf{occ}(g,Q)$) the number of (positive and negative) occurrences of $g$ in $P$ (respectively, $Q$). The required matching $\mathcal{M}$ thus needs to satisfy the following rule: for any gene $g$ in $P$ (respectively, $Q$), $\mathcal{M}$ must contain $\min\{\mathsf{occ}(g, P), \mathsf{occ}(g, Q)\}$ one-to-one correspondences involving $g$. In that sense, $\mathcal{M}$ remains a *full matching*, because it contains the maximum possible number of one-to-one correspondences between genes. The difficulty here is the following: because genomes are not balanced, some genes will remain unmatched by $\mathcal{M}$. In that case, we *prune* the genomes (*i.e.*, we remove those unmatched genes), to obtain two genomes $P'$ and $Q'$ in which each gene is covered by $\mathcal{M}$. We then compute the number of breakpoints between $P'$ and $Q'$ resulting from the permutation induced by $\mathcal{M}$.

We note, for the sake of completeness, that other types of matching exist that can be required:

- One can ask for an *exemplar* matching in which we only keep one occurrence of each gene $g$ [11]
- One also can ask for an *intermediate* matching in which for each gene $g$, we keep $1 \leq x \leq \min\{\mathsf{occ}(g, P), \mathsf{occ}(g, Q)\}$ occurrences of $g$ [1]

Coming back to the *full* matching variant, in both the balanced and unbalanced cases, finding a full matching that optimizes a given measure is NP-hard, and even APX-hard for all classical measures. This is, for instance, the case for minimizing the number of breakpoints [8], which maximizing the number of common intervals [2] or the number of conserved intervals [2], and this hardness holds even for very restricted instances. Consequently, most efforts in the literature have focused on what seemed to be the "simplest" case (*i.e.*, minimizing the number of breakpoints).

In the rest of this section, we describe two algorithms that deal with genomes $P$ and $Q$, represented as strings of integers, and aim to find a full matching $\mathcal{M}$ that minimizes the number of breakpoints in the permutation induced by $\mathcal{M}$ (possibly obtained after pruning in case $P$ and $Q$ are not balanced). Let us denote this problem by BAL-FMB (for Full Matching Breakpoints) ($P$, $Q$) in the balanced case, and UNBAL-FMB($P$, $Q$) in the unbalanced case. The two algorithms we describe here are:

1. An approximation algorithm for BAL-FMB [9]
2. An exact (thus, exponential) algorithm for UNBAL-FMB [1], written in the form of a 0–1 linear program, the goal being to be able to handle large instances

We want to emphasize the fact that this section does not aim to be an exhaustive survey of the results concerning BAL-FMB and UNBAL-FMB but to provide different algorithmic techniques and results that we think can be of interest for the reader.

### 34.3.2 An Approximation Algorithm for BAL-FMB

In this section, we show the main ideas and arguments of an approximation algorithm provided by Kolman and Waleń [9]. Let $P$ and $Q$ be two balanced strings containing signed integers, let $n = m_P = m_Q$, and let $k$ be the maximum number of copies of a gene in $P$ (respectively, in $Q$). Note that, because a gene is represented by a signed integer, $k$ takes into account both positive and negative occurrences of the most represented integer in $P$ (respectively, $Q$). The result from Kolman and Waleń [9] that we develop here is as follows.

**Theorem 34.1** *An $O(k)$ approximation algorithm exists for solving the problem* BAL-FMB.

***34.3.2.1 A Slightly Different Problem: Unsigned Minimum Common String Partition*** (UMCSP)**.** To make things simpler, we first develop the main arguments for the previous theorem in the specific case where the strings are *unsigned* (*i.e.*, every integer in both strings carries the same sign, which we always will consider as positive). The algorithm can be adapted easily for instances containing signed strings but with a loss of a factor two in the approximation ratio, which of course, does not change the ratio of $O(k)$ given in Theorem 34.1.

It should be said first that the result from Kolman and Waleń [9] considers a slightly different problem than BAL-FMB, called unsigned minimum common string partition (UMCSP). This problem is the following: given two balanced unsigned strings $P$ and $Q$, find a partition $\mathcal{P} = \{P_1, P_2, \dots P_t\}$ of substrings (*i.e.*, sets of consecutive elements) of $P$ such that:

1. $P_1 \cdot P_2 \cdot P_3 \dots P_t = P$, where $X \cdot Y$ denotes the concatenation of strings $X$ and $Y$
2. A permutation $\pi$ exists on $\{1, 2, \dots t\}$ such that $Q = P_{\pi(1)} \cdot P_{\pi(2)} \cdot P_{\pi(3)} \dots P_{\pi(t)}$
3. $t$ is minimized

In this context, each $P_i$, $1 \leq i \leq t$ is called a *block*. It is shown that BAL-FMB and UMCSP are related closely in the following sense:

- Any block partition of $P$ and $Q$ returned by UMCSP can be converted as a full matching between genes of $P$ and genes of $Q$ and vice-versa.
- If $b$ (respectively $t$) denotes the minimum number of breakpoints obtained by BAL-FMB (respectively the minimum number of blocks obtained by UMCSP), then $b$ and $t$ differ by one. Thus, any approximation algorithm of ratio $O(k)$ for UMCSP is also an approximation algorithm of ratio $O(k)$ for BAL-FMB.

As a consequence, in the rest of the section, we only focus on UMCSP, keeping in mind that the result also applies to BAL-FMB.

Before going into further details, we need a few definitions: a *substring* of a string $S$ is a set of consecutive elements of $S$. Recall that a *duo* in a string $S$ is just a substring of length two of $S$, and let us denote by duos($S$) the set of duos of $S$. Finally, given any solution for UMCSP, if a duo $d$ from $P$ does not appear in $\mathcal{P}$, then we say that $d$ is *broken*.

***34.3.2.2 A First Approximation Algorithm for* UMCSP*.*** The crucial idea behind the approximation algorithm from [9] is expressed as follows: in any solution for UMCSP, whenever a substring $X$ appears a different number of times in $P$ than in $Q$, at least one duo in at least one occurrence of $X$ must be broken. For any nonempty string $X$, we denote by #substr($P$, $X$) (respectively #substr($Q$, $X$)) the number of times $X$ appears as a substring of $P$ (respectively $Q$). Hence, the approximation algorithm ApproxUMCSP we look for could work as follows: for every $X$ such that #substr($P$, $X$) $\neq$ #substr($Q$, $X$), cut at least one duo in each occurrence of $X$ in $P$ and $Q$, and return the partitions $\mathcal{P}$ and $\mathcal{Q}$ induced by those cuts. The correctness of ApproxUMCSP is given by the following lemma.

**Lemma 34.4** *Algorithm* ApproxUMCSP *returns two partitions $\mathcal{P}$ and $\mathcal{Q}$ that form a common partition of $P$ and $Q$.*

Note that we should try to avoid too many cuts of duos because each cut of a duo corresponds to an increase in the number of blocks in $\mathcal{P}$ and $\mathcal{Q}$. However, minimizing the number of cuts is equivalent to the hitting set problem, which is known to be hard to approximate [10]. Thus, a deeper analysis is needed. Let $T$ denote the set of all substrings $X \in \Sigma^*$ such that #substr($P$, $X$) $\neq$ #substr($Q$, $X$). Then, it can be seen that not all substrings $X \in T$ need to be considered. Indeed, if two substrings $X, Y \in T$ are such that $X \sqsubset Y$ (where $X \sqsubset Y$ here means "$X$ is a proper substring of $Y$"), then any duo $d$ that breaks an occurrence of $X$ contained in $Y$ also will break $Y$. Thus, we can limit ourselves to the study of the set $T_{\min}$, which is defined as follows:

$$T_{\min} = \{X \in T | \nexists X' \in T \ s.t. \ X' \sqsubset X\}$$

In other words, $T_{\min}$ is the set of the substrings of $T$ that are minimal with respect to the relation $\sqsubset$.

Thus, instead of going through $T$, going through $T_{\min}$ only in algorithm ApproxUMCSP maintains its correctness. Now, to determine the approximation ratio we look for, we need to analyze how $T_{\min}$ is involved in any optimal solution of UMCSP. Let the two partitions $(\mathcal{P}_O, \mathcal{Q}_O)$ represent an optimal solution of UMCSP, and let an *optimal break* be any broken duo in this solution. The following lemma holds.

**Lemma 34.5** *If $X \in T_{min}$, then at least one occurrence of $X$ in $P$ or $Q$ exists that contains an optimal break.*

Our goal now is to assign to each $X \in T_{\min}$ an optimal break. For this, for any $X \in T_{\min}$, we denote by $f(X)$ the optimal break that $X$ contains. If $X$ contains more than one optimal break, then $f(X)$ is set arbitrarily to the leftmost one. In that case, the following lemma holds.

**Lemma 34.6** *Let $X = X[1]X[2]\ldots X[l]$ and $Y$ be two strings from $T_{\min}$ such that $f(X) = f(Y)$. In that case, $duos(Y) \cap \{X[1]X[2], X[l-1]X[l]\} \neq \emptyset$.*

Lemma 34.6 leads us directly to the following modification of algorithm ApproxUMCSP: for each $X \in T_{\min}$, we cut the first and last duo in all occurrences of $X$ in $P$ and $Q$. Algorithm ApproxUMCSP is now complete and is summarized in Algorithm 34.7.

**Algorithm 34.7** Algorithm ApproxUMCSP [10]

```
Input: Two balanced unsigned strings of integers, $P$ and $Q$,
each of length $n = m_P = m_Q$

Compute the set $T_{\min}$ defined in the text above
$\Phi = \emptyset$
$\mathcal{P} = \{P\}$, $\mathcal{Q} = \{Q\}$
For each $X \in T_{\min}$ do
    If $duos(X) \cap \Phi = \emptyset$ then
        Add the first and last duo of $X$ in $\Phi$
        Cut all occurrences of those two duos in the partitions $\mathcal{P}$ and $\mathcal{Q}$
    End If
End For

Output: Partitions $\mathcal{P}$ and $\mathcal{Q}$
```

It can be seen that Algorithm ApproxUMCSP remains correct because each occurrence of any $X \in T_{\min}$ is cut by at least one duo, and Lemma 34.4 still holds. Moreover, the following theorem holds.

**Theorem 34.2** Algorithm ApproxUMCSP is a $4k$ approximation algorithm for UMCSP.

The proof is as follows: suppose that $X_1$ and $X_2$ are two distinct strings of $T_{\min}$ that contributed to increasing the cardinality of the set $\Phi$ during the execution of ApproxUMCSP. Then, by Lemma 34.6, $f(X_1) \neq f(X_2)$. Because, on the whole, there are $\text{Card}(\mathcal{P}_O) + \text{Card}(\mathcal{Q}_O) - 2$ optimal breaks, this means that $\text{Card}(\Phi) \leq 2\,\text{Card}(\mathcal{P}_O) + 2\,\text{Card}(\mathcal{Q}_O) - 4$. Here, we consider instances in which a given integer appears at most $k$ times; thus, each duo from $\Phi$ induces at most $k$ cuts. Let $\mathcal{P}$ and $\mathcal{Q}$ be the partitions returned by ApproxUMCSP, and recall that, by definition, $\text{Card}(\mathcal{P}) = \text{Card}(\mathcal{Q})$ and $\text{Card}(\mathcal{P}_O) = \text{Card}(\mathcal{Q}_O)$. We have $\text{Card}(\mathcal{P}) \leq k\,\text{Card}(\Phi) + 1$; thus $\text{Card}(\mathcal{P}) \leq 4k(\text{Card}(\mathcal{P}_O) - 1) + 1$, that is, $\text{Card}(\mathcal{P}) \leq 4k\,\text{Card}(\mathcal{P}_O)$.

#### *34.3.2.3 About the Time Complexity of* ApproxUMCSP*.*

Kolman and Waleń have presented different tricks for achieving a time complexity of $O(n)$ for ApproxUMCSP, where $n = m_P = m_Q$.

First, instead of computing $T_{\min}$, it is sufficient to compute a set $T'$ of strings satisfying the three following properties:

1. $\text{Card}(T')$ is in $O(n)$ and can be computed in $O(n)$ time
2. $T_{\min} \subseteq T' \subseteq T$
3. If a string $X \in T$ passes the test of Line 6. of algorithm ApproxUMCSP (*i.e.*, $duos(X) \cap \Phi = \emptyset$), then $X \in T_{\min}$

In other words, $T'$ is just a set that is easier to compute than $T_{\min}$. Property 1 ensures it is not too large and that it can be found efficiently. Properties 2 and 3 ensure that ApproxUMCSP remains correct using $T'$ instead of $T_{\min}$.

$T'$ actually can be computed in $O(n)$ time using a suffix tree; let $P$ and $Q$ be the two balanced genomes from the instance. Then we build the (compact) suffix tree $\mathcal{T}$ of string $S = P\$_P Q\$_Q$, where $\$_P$ and $\$_Q$ are characters not appearing in $P$ and $Q$. Such a suffix tree can be constructed in $O(n)$ time [13]. Let $r$ be the root of $\mathcal{T}$, and let $v \neq r$ be any node of $\mathcal{T}$. Let parent($v$) be the father of $v$ in $\mathcal{T}$, let $s(v)$ be the string represented by the path from $r$ to parent($v$), and let $s'(v) = s(v) \cdot c$, where $c$ is the first character of the string represented by the edge {parent($v$), $v$}. Finally, we say that $v$ is a *proper* node of $\mathcal{T}$ when $s'(v)$ contains neither $\$_P$ or $\$_Q$. Now we can define $T'$; $T'$ is the set of the strings $s'(v)$ for any proper node $v$ in $\mathcal{T}$ for which $\#\text{substr}(P, s'(v)) \neq \#\text{substr}(Q, s'(v))$. One can see that $\text{Card}(T')$ is in $O(n)$ because $\mathcal{T}$ contains $O(n)$ nodes. Besides, $T_{\min} \subseteq T'$ by definition. Finally, using the suffix tree, $\#\text{substr}(P, s'(v))$ (respectively $\#\text{substr}(Q, s'(v))$) can be computed in $O(n)$ time, and $T'$ thus can be computed in $O(n)$ time as well.

Second, Kolman and Waleń describe how to maintain the set $\Phi$, test the condition of Line 6 of algorithm ApproxUMCSP, and realize the cuts in $O(1)$ time, leading to $O(n)$ overall. For this, they use a data structure from Gabow and Tarjan [7] that ensures an amortized $O(1)$ time for each such operation.

#### *34.3.2.4 From* UMCSP *to the Signed Case* SMCSP*.*

The previous description was focused on UMCSP, that is, the unsigned case. If we want to adapt ApproxUMCSP to the signed case (a problem that we call SMCSP), then a few adaptations need to be made. For any string $S$, let $-S$ denote its reversed string, both in order and sign. Then three main changes need to be done:

- #substr$(P, X)$ now should count the number of occurrences of $X$ and $-X$ in $P$
- The set $T'$ should be computed using the suffix tree $\mathcal{T}'$ of string $S' = P\$_P Q\$_Q(-P)\$_P(-Q)\$_Q$ (where the brackets are here just to delimit strings)
- Whenever a duo $ab$ should be cut, all duos $-b - a$ should be cut as well

None of these adaptations changes the time complexity or the correctness of the algorithm. However, the last one increases the approximation ratio of ApproxUMCSP by a factor of two.

***34.3.2.5 Remarks.*** First, we note that approximation algorithm ApproxUMCSP is actually a $\Theta(k)$ approximation algorithm because instances exist for which the optimum number of blocks is $O(1)$, whereas the number of blocks returned by the algorithm is $O(k)$. Strings $P = ba\{ab\}^{k-1}$ and $Q = ab^k$ [9] form such an instance; a partition of $P$ (respectively $Q$) exists containing three blocks, but ApproxUMCSP will return a solution containing $k + 1$ blocks.

It also should be noted that, strangely enough, if instead of trying to minimize the number of breakpoints, we aim to find a full matching that *maximizes the number of adjacencies*, then a four-approximation algorithm exists [2] (*i.e.*, an approximation ratio that does not depend on $k$). However, each problem is, in some sense, the dual of the other. This raises the question of whether we can do better than an $O(k)$-approximation algorithm for `BAL-FMB`, and more precisely, is `BAL-FMB` $O(1)$-approximable?

### 34.3.3 An Exact Algorithm for `UNBAL-FMB`

In this section, we focus on the problem `UNBAL-FMB` in which the goal is to find a full matching $\mathcal{M}$ between two *unbalanced* signed genomes in such a way that the permutation induced by $\mathcal{M}$ minimizes the number of breakpoints.

Here, we give the main elements of an exact algorithm that solves `UNBAL-FMB`, published in [1]. The problem is known to be APX-hard, even for instances in which $P$ does not contain duplicates and occ(g,Q)$\leq$ 2 for any gene $g$ from $Q$ [2]. Thus, the algorithm we give here is exponential; our approach is to express `UNBAL-FMB` in a 0–1 linear program, that is, a series of inequalities implying Boolean variables only, together with an objective function on Boolean variables, that we wish to maximize.

The main interest in such an approach lies in the fact that there has been many efforts in the past to develop software that can handle such programs even if they contain a large number of variables and inequalities. Thus, our hope is that powerful enough solvers (such as minisat+ or CPLEX) can provide optimal solutions on real data in a reasonable time. If this is the case, then we have two options:

- We can solve exactly those instances for themselves. However, even if this works for some data, one easily can imagine that instances exist that never will be solved in reasonable time.

- Thanks to the exact results obtained by this method, we can analyze and evaluate one or several heuristic(s), as was done, for example, in [1].

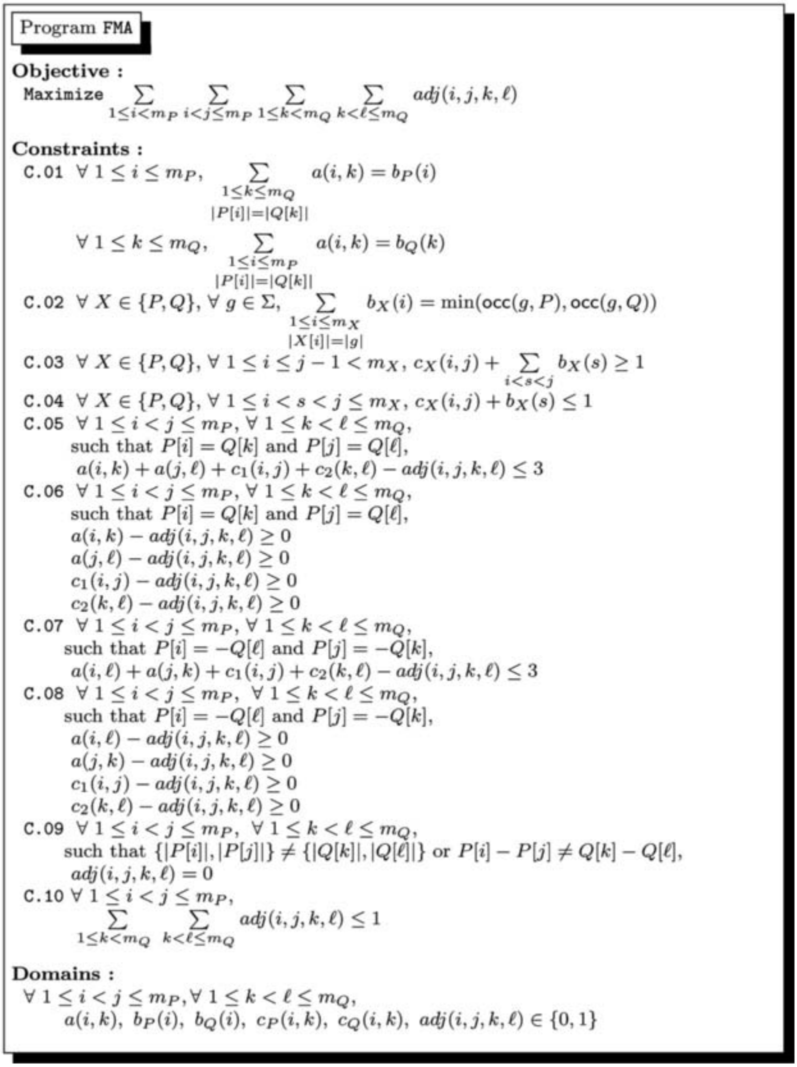
As we will see later, we have tested our program on a set of 12 genomes of bacteria, for which all 66 pairwise comparisons were achieved rapidly.

We first present below the complete 0–1 linear program in itself (full matching adjacencies [FMA], a name that will be justified later), together with an explanation of the different variables we have defined and used. Next, a few data reduction rules are provided that aim to reduce the input size, and hence to speedup the program.

Program FMA takes as input two genomes $P$ and $Q$ with duplicates, of respective lengths $m_P$ and $m_Q$, and solves problem UNBAL-FMB. Recall that $\Sigma$ denotes the set of integers (representing genes) on which $P$ and $Q$ have been built. The objective function, the variables and the constraints involved now are discussed (see also Figure 34.2).

### *34.3.3.1 Variables*

- Variables $adj(i, j, k, \ell)$, $1 \leq i < j \leq m_P$ and $1 \leq k < \ell \leq m_Q$, represent *adjacencies* according to $\mathcal{M}$. Our initial problem is to try to minimize the number of breakpoints; however, maximizing the number of adjacencies makes the writing of our 0–1 linear program more simple. Besides, it can been seen easily that because Card($\mathcal{M}$) is given by the input, the full matching that minimizes the number of breakpoints also maximizes the number of adjacencies. Thus, we only focus on adjacencies here, and $adj(i, j, k, \ell) = 1$ if and only if the three following properties are satisfied:
  1. One of the two following cases occur:
     - $(P[i], Q[k])$ and $(P[j], Q[\ell])$ belong to $\mathcal{M}$, $P[i] = Q[k]$ and $P[j] = Q[\ell]$
     - $(P[i], Q[\ell])$ and $(P[j], Q[k])$ belong to $\mathcal{M}$, $P[i] = -Q[\ell]$ and $P[j] = -Q[k]$
  2. $P[i]$ and $P[j]$ are consecutive in $P$ according to $\mathcal{M}$
  3. $Q[k]$ and $Q[\ell]$ are consecutive in $Q$ according to $\mathcal{M}$
- Variables $a(i, k)$, $1 \leq i \leq m_P$ and $1 \leq k \leq m_Q$, define a matching $\mathcal{M}$: $a_{i,k} = 1$ if and only if $P[i]$ is matched with $Q[k]$ in $\mathcal{M}$.
- Variables $b_X(i)$, $X \in \{P, Q\}$ and $1 \leq i \leq m_X$, define whether the gene appearing at position $i$ of $X$ is covered by the matching $\mathcal{M}$. More precisely, $b_X(i) = 1$ if and only if $X[i]$ is covered by $\mathcal{M}$. Clearly, $\sum_{1 \leq i \leq m_P} b_P(i) = \sum_{1 \leq k \leq m_Q} b_Q(k)$, and this is precisely the size of $\mathcal{M}$.
- Variables $c_X(i, j)$, $X \in \{P, Q\}$ and $1 \leq i < j \leq m_X$, determine whether genes at positions $i$ and $j$ in $X$ are *consecutive genes* according to $\mathcal{M}$: $c_X(i, j) = 1$ if and only if $X[i]$ and $X[j]$ both are covered by $\mathcal{M}$, and no gene $X[p]$, $i < p < j$, is covered by $\mathcal{M}$.



Program FMA

**Objective :**

Maximize $\sum_{1\le i<m_P}\ \sum_{i<j\le m_P}\ \sum_{1\le k<m_Q}\ \sum_{k<\ell\le m_Q} adj(i,j,k,\ell)$

**Constraints :**

C.01 $\forall\, 1 \le i \le m_P,\ \sum_{\substack{1\le k\le m_Q\\ |P[i]|=|Q[k]|}} a(i,k) = b_P(i)$

$\forall\, 1 \le k \le m_Q,\ \sum_{\substack{1\le i\le m_P\\ |P[i]|=|Q[k]|}} a(i,k) = b_Q(k)$

C.02 $\forall\, X \in \{P,Q\}, \forall\, g \in \Sigma,\ \sum_{\substack{1\le i\le m_X\\ |X[i]|=|g|}} b_X(i) = \min(\mathsf{occ}(g,P), \mathsf{occ}(g,Q))$

C.03 $\forall\, X \in \{P,Q\}, \forall\, 1 \le i \le j-1 < m_X,\ c_X(i,j) + \sum_{i<s<j} b_X(s) \ge 1$

C.04 $\forall\, X \in \{P,Q\}, \forall\, 1 \le i < s < j \le m_X,\ c_X(i,j) + b_X(s) \le 1$

C.05 $\forall\, 1 \le i < j \le m_P, \forall\, 1 \le k < \ell \le m_Q$,
such that $P[i] = Q[k]$ and $P[j] = Q[\ell]$,
$a(i,k) + a(j,\ell) + c_1(i,j) + c_2(k,\ell) - adj(i,j,k,\ell) \le 3$

C.06 $\forall\, 1 \le i < j \le m_P, \forall\, 1 \le k < \ell \le m_Q$,
such that $P[i] = Q[k]$ and $P[j] = Q[\ell]$,
$a(i,k) - adj(i,j,k,\ell) \ge 0$
$a(j,\ell) - adj(i,j,k,\ell) \ge 0$
$c_1(i,j) - adj(i,j,k,\ell) \ge 0$
$c_2(k,\ell) - adj(i,j,k,\ell) \ge 0$

C.07 $\forall\, 1 \le i < j \le m_P, \forall\, 1 \le k < \ell \le m_Q$,
such that $P[i] = -Q[\ell]$ and $P[j] = -Q[k]$,
$a(i,\ell) + a(j,k) + c_1(i,j) + c_2(k,\ell) - adj(i,j,k,\ell) \le 3$

C.08 $\forall\, 1 \le i < j \le m_P,\ \forall\, 1 \le k < \ell \le m_Q$,
such that $P[i] = -Q[\ell]$ and $P[j] = -Q[k]$,
$a(i,\ell) - adj(i,j,k,\ell) \ge 0$
$a(j,k) - adj(i,j,k,\ell) \ge 0$
$c_1(i,j) - adj(i,j,k,\ell) \ge 0$
$c_2(k,\ell) - adj(i,j,k,\ell) \ge 0$

C.09 $\forall\, 1 \le i < j \le m_P,\ \forall\, 1 \le k < \ell \le m_Q$,
such that $\{|P[i]|, |P[j]|\} \ne \{|Q[k]|, |Q[\ell]|\}$ or $P[i] - P[j] \ne Q[k] - Q[\ell]$,
$adj(i,j,k,\ell) = 0$

C.10 $\forall\, 1 \le i < j \le m_P$,
$\sum_{1\le k<m_Q}\ \sum_{k<\ell\le m_Q} adj(i,j,k,\ell) \le 1$

**Domains :**

$\forall\, 1 \le i < j \le m_P, \forall\, 1 \le k < \ell \le m_Q$,
$a(i,k),\ b_P(i),\ b_Q(i),\ c_P(i,k),\ c_Q(i,k),\ adj(i,j,k,\ell) \in \{0,1\}$

**Figure 34.2** Program FMA solves exactly UNBAL-FMB.

#### *34.3.3.2 Constraints.* Assume $1 \le i < j \le m_P$ and $1 \le k < \ell \le m_Q$.

- Constraint C.01 ensures that each gene of $P$ and of $Q$ is matched at most once (*i.e.*, $b_P(i) = 1$ (respectively $b_Q(k) = 1$) if and only if gene $i$ (respectively $k$) is matched in $P$ (respectively $Q$)). Observe that in any matching, any two genes that are mapped together necessarily have the same label (except maybe for the sign), and hence, we do not have to ask explicitly for $a(i,k) = 0$ in case $P[i]$ and $Q[k]$ are two different genes.

- Constraint C.02 actually defines the fact that we ask for a full matching. For each gene $g$, $\min(\text{occ}(g, P), \text{occ}(g, Q))$ occurrences of $g$ must be covered by $\mathcal{M}$ in both $P$ and $Q$.
- Constraints in C.03 and C.04 are concerned with our definition of consecutive genes. Variable $c_X(i, j)$ is equal to 1 if and only if no $p$ exists such that $i < p < j$ and $b_X(p) = 1$. It is worth noticing here that, according to these constraints, one may have $c_X(i, j) = 1$ even if one of the genes $X[i]$ or $X[j]$ is *not* covered by $\mathcal{M}$.
- Constraints in C.05 to C.10 define variables adj. In the case where $P[i] = Q[k]$ and $P[j] = Q[\ell]$, constraints C.05 and C.06 ensure that we have $\text{adj}(i, j, k, \ell) = 1$ if and only if all variables $a(i, k)$, $a(j, \ell)$, $c_1(i, j)$, and $c_2(k, \ell)$ are equal to 1. In the case where $P[i] = -Q[\ell]$ and $P[j] = -Q[k]$, Constraints C.07 and C.08 ensure that we have $\text{adj}(i, j, k, \ell) = 1$ if and only if all variables $a(i, \ell)$, $a(j, k)$, $c_1(i, j)$, and $c_2(k, \ell)$ are equal to 1. Constraint C.09 sets variable $\text{adj}(i, j, k, \ell)$ to 0 if none of these two cases hold. Finally, thanks to constraint C.10, one must have at most one adjacency for every pair $(i, j)$.

The objective of Program FMA is to maximize the number of adjacencies between the two considered genomes. According to the above data, this objective thus reduces in our model to maximizing the sum of all variables $\text{adj}(i, j, k, \ell)$.

***34.3.3.3 Speeding Up the Program.*** Program FMA has $O((m_P\, m_Q)^2)$ variables and $O((m_P\, m_Q)^2)$ constraints. To speed up the execution of the program, there are some simple rules to apply for reducing the number of variables and constraints in FMA.

First, the genomes are pairwise preprocessed to delete all genes that do not appear in both genomes because we know that no full matching will contain them.

Second, for any gene $g$ for which $\text{occ}(g, P) = \text{occ}(g, Q) = 1$ and $|P[i]| = |Q[k]| = g$, the corresponding variable $a_{i,k}$ is set directly to 1 as well as the two variables $b_P(i)$ and $b_Q(k)$.

Also, if two genes appearing only once in each genome occur consecutively or in reverse order with opposite signs, the corresponding variable adj is set directly to 1, and the related constraints are discarded.

Finally, if for two genes, say occurring at positions $i$ and $j$ in $P$, at least one gene $g$ occurring between position $i$ and $j$ in $P$ must be covered in any matching $\mathcal{M}$ (*e.g.*, if all occurrences of $g$ appear between $i$ and $j$ in $P$), then the corresponding variable $c_P(i, j)$ and the variables $\text{adj}(i, j, k, \ell)$ for all $1 \leq k < \ell \leq m_Q$ are set directly to 0, and the related constraints are discarded. Of course, the same reasoning applies for two positions $k$ and $\ell$ in $Q$ and the variables $c_Q(k, \ell)$ and $\text{adj}(i, j, k, \ell)$ for all $1 \leq i < j \leq m_P$.

***34.3.3.4 Remarks.*** FMA has been tested on 12 genomes of $\gamma$-*Proteobacteria* (a subfamily of bacteria), which contain from 564 to 5540 genes (3104 on average).

This led to 66 pairwise comparison that were achieved within two minutes (leading to an average of 1.7 s per comparison) using the solver CPLEX [1].

FMA works for unbalanced genomes and can be simplified greatly to be adapted to balanced ones; more precisely, some variables and thus some constraints do not need to exist anymore. For instance, if $P$ and $Q$ are balanced and of length $n$, $b_P(i)$ (respectively $b_Q(i)$) is set to 1 for any $1 \leq i \leq n$, and $c_P(i, j)$ (respectively $c_Q(i, j)$) are unnecessary. The same goes for some constraints such as (C.03) and (C.04).

### 34.3.4 Other Results and Open Problems

***34.3.4.1 Balanced Case.*** Apart from the approximation algorithm given in Section 34.3.2, the main recent result is a fixed-parameter tractable (FPT) algorithm for UMCSP by Damaschke [5]. More precisely, the main result from [5] is the following: an FPT algorithm for UMCSP exists on $P$ and $Q$, whose exponential running time involves only parameters $b$ and $r$, where:

- $b$ is the minimum number of blocks in an optimal solution for UMCSP on $P$ and $Q$
- $r$ is the *repetition number* of $P$, that is, the maximum $i$ such that $P = X \cdot Y^i \cdot Z$ for some strings $X$, $Y$, and $Z$, where $Y$ is nonempty.

Two main open problems remain:

1. Does an approximation algorithm exist of ratio $O(1)$ for SMCSP?
2. Is UMCSP (respectively SMCSP) fixed-parameter tractable on $b$ only?

***34.3.4.2 Unbalanced Case.*** In this case, to our knowledge, no positive result (polynomial time approximation scheme, approximation algorithm or FPT algorithm) is known for UNBAL-FMB, even for restricted cases. In that sense, the field is totally open.

We note though, that a related problem, called Zero (full) Matching Breakpoint Distance (ZMBD), has been shown to be polynomial in [2]. ZMBD is the following decision problem: given two signed unbalanced genomes, determine whether a full matching $\mathcal{M}$ exists such that the number of breakpoints in the permutation induced by $\mathcal{M}$ is equal to zero.

## 34.4 CONCLUSION

In this chapter, we have presented different algorithmic techniques for comparing pairs of genomes to infer (dis)similarity measures between them. The two main types of measures that have been studied were: (i) common and conserved intervals in the first part and (ii) breakpoints and adjacencies in the second part.

We intentionally did not provide a survey of all existing results on the topic, but we have chosen to focus only on a few algorithms to show the different techniques and ideas that lie behind those algorithms. We think and hope this could be of interest for the reader. It also allowed us to show a sample of the ideas that have been developed lately in the algorithmic field of comparative genomics.

As is shown in this chapter, all aforementioned measures can be computed in polynomial time whenever genomes are permutations. On the contrary, when genomes contain duplicates, and in case a matching is required, all measures are hard to compute and even hard to approximate, even in very restricted cases. It is shown, however, that when genomes are balanced, some positive results exist in the form of approximation and FPT algorithms in the full matching case.

The most challenging questions that remain open in this domain are probably those that ask for positive results for comparing *unbalanced* genomes using a matching and any of the aforementioned measures.

## REFERENCES

1. S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. *J Comput Biol*, 15(8):1093–1115, 2008.
2. S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. On the approximability of comparing genomes with duplicates. *J Graph Algorithm Appl*, 13(1):19–53, 2009.
3. A. Bergeron, C. Chauve, F. de Montgolfier, and M. Raffinot. Computing common intervals of $K$ permutations, with applications to modular decomposition of graphs. *SIAM J Discrete Math*, 22(3):1022–1039, 2008.
4. A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. *J Comput Biol*, 13(7):1340–1354, 2006.
5. P. Damaschke. Minimum common string partition parameterized. *Proceedings of WABI 2008*, Volume 5251 of Lecture Notes of Computer Science, Springer, New York, 2008, pp. 87–98.
6. G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. MIT Press, Cambridge, MA, 2009.
7. H.N. Gabow and R.E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J Comput Syst Sci*, 30(2):209–221, 1985.
8. A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. *Electron J Combinator* 12(1):paper R50, 2005.
9. P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electron J Combinator*, 14(1):R50, 2007.
10. R. Raz and S. Safra. A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. *Proceedings of STOC 97*, ACM, Mountain View, CA, 1997, 475–484.
11. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

12. T. Schmidt and J. Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. *Proceedings CPM 2004*, Volume 3109 of Lecture Notes in Computer Science, Springer, New York, 2004, pp. 347–358.
13. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
14. T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26:290–309, 2000.

# 35

# WAVELET ALGORITHMS FOR DNA ANALYSIS

Carlo Cattani

## 35.1 INTRODUCTION

One of the main tasks of the genome project is to understand completely the underlying biological function from a possible interpretation of the given sequence of nucleotides that is from the distribution of the four symbols A, C, G, T along the sequence [21, 24, 25]. The main hypotheses of this project are as follows:

1. The activity (functional) of the organism is a result of the distribution of nucleotides.
2. The distribution of nucleotides should follow some hidden rules.
3. It should be possible to discover these rules by singling out some regular features like periodicity, typical patterns, trends, sequence evolution, and so on.

In recent years, the analysis of DNA sequences has been focused mainly on the existence of hidden law, periodicities, and autocorrelation [14, 17, 24, 34]. The main task is to find (if any) some kind of mathematical rules or meaningful statistics in the nucleotides distribution. This would help us to characterize each DNA sequence to construct a possible classification. From a mathematical point a view, the DNA sequence is a symbolic sequence (of nucleotides) with some empty spaces (no coding regions). To get some numerical information from this sequence, it must be transformed into a digital sequence. It follows that the symbolic sequence is transformed into a very large time series (from one half of a million digits for the primitive

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

organisms such as fungus, eukaryotes, to several million as for mammals, like the nearly 1.5 billion nucleotides for human DNA). These large sequences look like some random sequence, from which it seems to be impossible to single out any single correlation (see *e.g.*, [17] and references therein).

The DNA sequence consists of a large string made by four chemical elements (nucleotides) called bases (or base pairs): adenine (A), cytosine (C), guanine (G), and thymine (T). They are combined in a such way to form a long filament that has the structure of a double spiral, which is a very steady chemical structure. When the symbolic sequence of A , C , G , T is digitalized into one or more sequences of digits, one may benefit from the statistical analysis of the digitalized time series (signal) so that the genome can be characterized by the classical statistical parameters like variance, deviation, nonclassical-like complexity, fractal dimension, or long-range dependence.

In any case, one of the main problems is to assign one (or more) representative digital time series (discrete time signal) to the symbolic sequence of the genome so that the representation would be the most suitable for the statistical–mathematical analysis. However, the digitalization of a symbolic sequence must be done with care by avoiding some dependence or degeneracy of the representation. In other words, it is always possible that the analysis of the numerical signal could be dependent on the method used for the digitalization.

Some problems in DNA analysis are the understanding of the underlying genomic language to find an organization principle of the genome, to discover some kind of order (symmetries) or hidden structures (patches or regular patterns), and to understand the existence of functions on genes such as localized periodicities, correlation, complexity, and so on. The easiest mathematical model is based on the transformation of the symbolic string into a numerical string based on the Voss indicator function [40, 41], which is a discrete binary function. In the following, a complex representation is proposed to single out a fractal law in the cumulative distribution of nucleotides. The existence of patterns and symmetries is shown through the cluster analysis of the wavelet coefficients [12, 13].

Only recently, the analysis of DNA sequences has been improved by using wavelets (see, *e.g.*, [14, 34, 38] ). With wavelet analysis, it is possible to single out singularities, frequency content, fractal nature, and compression properties on DNA sequences [1, 4, 34, 38, 46]. This choice was motivated by the fundamental properties of wavelets, in fact,

1. With the localization property [14], it is possible (at least in principle) to single out local behavior and to characterize local spikes and jumps [12, 14].
2. Because of the decorrelation process of the wavelet transform [37], the DNA sequence is decomposed into sequences of detail coefficients at different levels, each one expressing some kind of autocorrelation on the corresponding level.

However, the wavelet transform of a sequence with a huge number of data (like the walks on DNA) gives a (huge) sequence of detail coefficients, which is meaningless when we want to focus on the existence of local (short) or long-range correlations. If we are interested on the jumps that can develop from one element of the series and the closer element of the sequence, then we must reduce the number of the elements to be mapped into the wavelet space. This can be achieved by a decomposition of the sequence into short segments of equal length and by a wavelet transform to be applied to each segment.

A fundamental problem in DNA analysis is whether a long-range correlation exits in the digital representation of the DNA sequence [3, 6, 10, 11, 29, 32, 33, 35, 36, 40, 41, 42, 45]. A correlation in a digital signal can be linked roughly with the concept of dependence, in a statistical sense, of elements that are far away from each other. Correlation in a DNA sequence is interesting because base pairs in a sequence of millions of pairs seem to have some statistical dependence. The existence of correlation in DNA has been explained with the so-called process of duplication-mutation. According to [32, 39], in the evolutionary model, the actual DNA sequence results from an original short-length chain that was duplicating and modifying some pieces of the sequence. Because of this, the characterizing $1/f$ power law decay followed [10, 11, 41].

The power law for long-range correlations is a measure of the scaling law, showing the existence of self-similar structures similar to the physics of fractals. The long-range correlation, which can be detected by the autocorrelation function, implies the scale independence (scale invariance), which is typical of fractals. The autocorrelation also is used for measuring linear dependence and periodicity. It has been applied to the analysis of DNA sequences [8, 31, 33, 35, 36, 40, 41] to prove the existence of scale invariance. However, the preliminary results in this topic were disputed [10, 11, 30] because of the limited number of available data and because of different approaches to this analysis. On the other hand, the existence of patchiness and correlation would imply some important understanding of DNA organization. It has been observed that the source for long-range correlation is linked with the existence of patchiness in the DNA sequence. The identification of these patches could be the key point for understanding the large-scale structure of DNA.

This chapter is organized as follows: Section 35.2 deals with some preliminary remarks on DNA and DNA representation. The indicator matrix is given, and its global fractal estimate is computed as well. In particular, the indicator matrix shows the existence of fractal patterns. Section 35.3 deals with some remarks on statistical parameters on the concept of long range correlation. The power spectrum is computed for candida's and dog's DNA, and the existence of long-range correlation is shown. Section 35.4 deals with the Haar wavelet theory. Statistical parameters, such as variance, fluctuation and the Hurst exponent are expressed in terms of wavelet coefficients in Section 35.5. The short Haar wavelet transform is given in Section 35.6. The cluster analysis of the complex representation and walk on dog's and candida's DNA are given in Section 35.7 and compared with clusters for white noise (random walks for pseudo random complex sequences). Also, in this case, the existence of long range correlation and, fractal behavior is shown. The possibility to detect easily anomalies in base pair distribution by clusters of wavelet coefficients is shown in some examples.

## 35.2 DNA REPRESENTATION

### 35.2.1 Preliminary Remarks on DNA

Genome is the whole set of genetic material (DNA) in the chromosomes of a living organism. The DNA of each organism of a given species is a long sequence of a specific (large) number of base pairs. The size of the DNA might range from $10^5$ to $10^9$ base pairs. Each base pair is defined on the four elements alphabet of nucleotides:

$$\mathrm{A} = \text{adenine}\ ,\ \mathrm{C} = \text{cytosine}\ ,\ \mathrm{G} = \text{guanine}\ ,\ \mathrm{T} = \text{thymine}$$

Because the base pairs are distributed along a double-helix, when straightened, the helix appears as a double strands system:

$$\xrightarrow{5'} \cdots \mathrm{A\ T\ T\ C\ A\ T\ A\ G} \cdots \xrightarrow{3'}$$

$$\xleftarrow{3'} \cdots \mathrm{T\ A\ A\ G\ T\ A\ T\ C} \cdots \xleftarrow{5'}$$

Both strands have to be read from $5'$ end to $3'$ end, left to right, for the upper strand, and vice-versa for the lower strand. The two sequences on opposite strands are complementary in the sense that opposite nucleotides must fulfill the ligand rules of base pairs

$$\mathrm{A} \longleftrightarrow \mathrm{T} \qquad \mathrm{C} \longleftrightarrow \mathrm{G}$$

In particular, A and G are called purines, whereas C and T are pyrimidines.

In a DNA sequence, there are some subsequences, coding and noncoding regions, with a special meaning. In particular, genes (coding regions) are characteristic sequences of base pairs, and the genes in turn are made by some alternating subsequences of exons and introns (except Procaryotes, which miss introns):

$$\mathrm{A\,T\,C\,G}\ldots\underbrace{\overbrace{\mathrm{A\,T}\ldots\mathrm{G\,A\,G}}^{gene}}_{\text{...exon...intron...exon...}}\ldots\mathrm{T\,C\,G}\ldots\underbrace{\overbrace{\mathrm{C\,A}\ldots\mathrm{T\,A\,G}}^{gene}}_{\text{...exon...intron...exon...}}\ldots$$

Each exon region is made of triplets of adjacent bases called codon. Because the bases are four, there are 64 possible codons. Each codon synthesizes a specific amino acid so that a sequence of codons defines a protein. A protein is a set of 20 different amino acids, and the amino acids are made by codons. There are only 20 proteins; therefore, the correspondence codons to proteins is many to one. The exons region also is called the coding region and because of the triplet formation, it exhibits a period-three behavior (on short term). So that by a discrete Fourier transform, there is a peak in correspondence of the frequency $2\pi/3$ (see, *e.g.*, [39]).

Let

$$\mathcal{A} \overset{\text{def}}{=} \{\mathrm{A}\ ,\mathrm{C}\ ,\mathrm{G}\ ,\mathrm{T}\}$$

be the finite set (alphabet) of nucleotides and $x \in \mathcal{A}$ be any member of the alphabet.

A DNA sequence is the finite symbolic sequence

$$\mathcal{S} = \mathbb{N} \times \mathcal{A}$$

so that

$$\mathcal{S} \overset{\text{def}}{=} \{x_h\}_{h=1,\ldots,N}\ , \quad N < \infty$$

being

$$x_h \overset{\text{def}}{=} (h, x) = x(h), \quad (h = 1, 2, \ldots, N;\ x \in \mathcal{A}) \tag{35.1}$$

the value $x$ at the position $h$.

### 35.2.2 Indicator Function

The indicator function [40] is the map

$$u : \mathcal{S} \times \mathcal{S} \to \{0\ , 1\}$$

such that for a fixed $x_h \in \mathcal{S}$

$$u_{x_h}(x_k) \overset{\text{def}}{=} u(x_h,\ x_k) = \begin{cases} 1 & \text{if}\ \ x_h = x_k \\ 0 & \text{if}\ \ x_h \neq x_k \end{cases} \quad (x_h \in \mathcal{S}\ ,\ x_k \in \mathcal{S}) \tag{35.2}$$

According to (35.2), the indicator of an $N$-length sequence easily can be represented by the $N \times N$ sparse matrix of binary values {0, 1}, and this matrix can be plotted in two dimensions by putting a dot where (Figures 35.1 and 35.2) $u_{hk} = 1$ and a white spot when $u_{hk} = 0$ being

$$u_{hk} \overset{\text{def}}{=} u_{x_h}(x_k) \qquad (x_h \in \mathcal{S}\ ,\ x_k \in \mathcal{S}; h, k = 0, \ldots, N - 1)$$

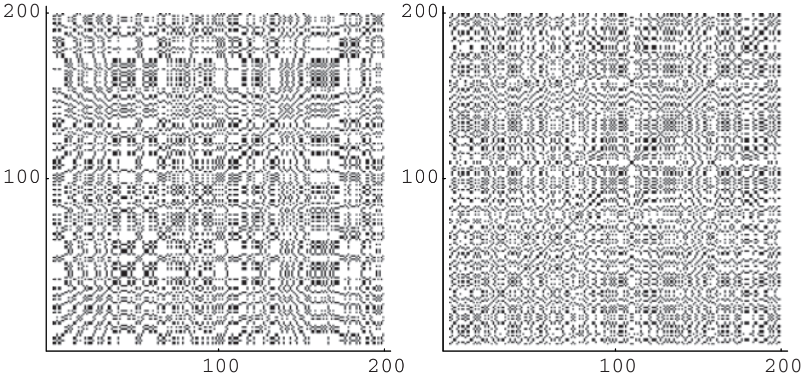


**Figure 35.1** Indicator matrix ($n \leq 200$) for dog's DNA (left) and candida's (right).

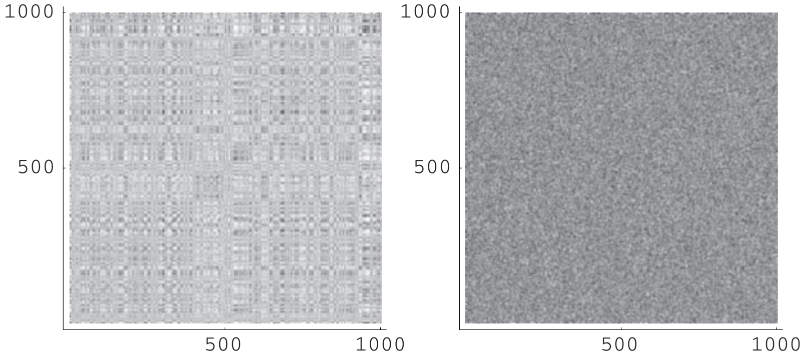


**Figure 35.2** Indicator matrix ($n \leq 1000$) for the dog's DNA (left) and a pseudorandom array (right).

which results from the following indicator table/matrix:

$$
\begin{array}{c|cccccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \;\; \cdot^{\cdot^{\cdot}} \\
\mathrm{G} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \ldots \\
\mathrm{C} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \ldots \\
\mathrm{A} & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \ldots \\
\mathrm{A} & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \ldots \\
\mathrm{T} & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \ldots \\
\mathrm{A} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \ldots \\
\mathrm{C} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \ldots \\
\mathrm{T} & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \ldots \\
\mathrm{G} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \ldots \\
\mathrm{A} & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \ldots \\
\hline
u_{hk} & \mathrm{A} & \mathrm{G} & \mathrm{T} & \mathrm{C} & \mathrm{A} & \mathrm{T} & \mathrm{A} & \mathrm{A} & \mathrm{C} & \mathrm{G} \ldots
\end{array}
$$

where both on the bottom and on the left there is the sequence $\mathcal{S}$, and the composition table is done according to the indicator values $u_{hk}$.

It is shown in Figure 35.1, and Figure 35.2 (left) that

1. Some motifs are repeated at different scales like in a fractal.
2. Empty spaces are more distributed than filled spaces in the sense that the matrix $u_{hk}$ is a sparse matrix (having more zeroes than ones).
3. It seems that there are some square-like islands where black spots are more concentrated.
4. The comparison of the DNA indicator matrix with a corresponding pseudorandom matrix clearly shows the existence of some hidden rule for the correlation of nucleotides.

From the indicator matrix, we can have an idea of the "fractal-like" distribution of nucleotides; however, there is no one-to-one correspondence between the indicator matrix and the DNA sequence. This can be realized by the barcode matrix, defined as follows:

$$v_{hk} \overset{\text{def}}{=} u_{x_h}(x_k) \qquad (x_h \in \mathcal{S}\ ,\ x_k \in \mathcal{A}; h = 0, \ldots, N-1, k = 1, \ldots, 4)$$

which results from the barcode table

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| T | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | ... |
| G | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ... |
| C | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | ... |
| A | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ... |
| $v_{hk}$ | A | G | T | C | A | T | A | A | C | G | ... |

where the composition table is done according to the indicator values $v_{hk}$. By putting a black bar where $v_{hk} = 1$ and a white bar when $u_{hk} = 0$, we obtain Figure 35.3.

Let

$$p_X(\ell), \qquad X \in \mathcal{A} = \{A,\ C,\ G,\ T\}$$

be the probability to find the nucleotide $X$ at the distance $\ell$; we can see that for higher values of $\ell$, the probabilities tend to assume some constant values thus showing that nucleotides are distributed heterogeneously. For the the dog's DNA, it is:

$$p_A = 0.36,\ \ p_C = 0.19,\ \ p_G = 0.19,\ \ p_T = 0.26 \qquad (35.3)$$

and for the candida's DNA, it is.

$$p_A = 0.32,\ \ p_C = 0.16,\ \ p_G = 0.17,\ \ p_T = 0.35 \qquad (35.4)$$

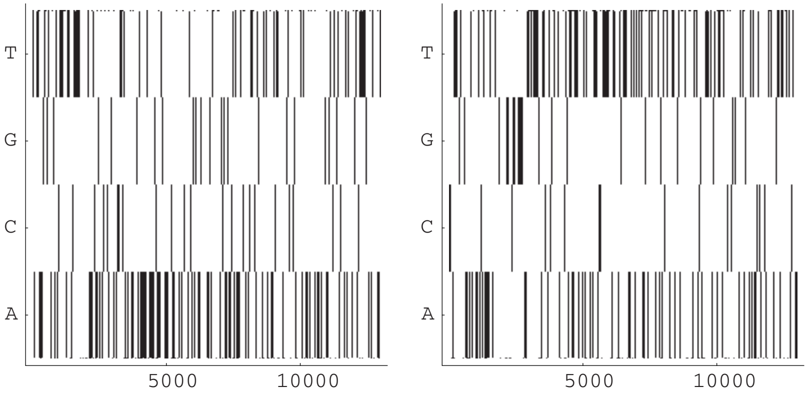


**Figure 35.3** Barcode matrix ($n \leq 15000$) for the dog's DNA (left) and the candida's (right).

***35.2.2.1 Global Fractal Estimate by the Correlation Matrix.*** By using the indicator matrix, it is possible to give a simple formula that enables us to estimate the fractal dimension as the average of the number $p(n)$ of one in the randomly taken $n \times n$ minors of the $N \times N$ correlation matrix $u_{hk}$

$$D = \frac{1}{N}\sum_{n=2}^{N}\frac{\log p(n)}{\log n} \tag{35.5}$$

The fractal dimension of the dog's DNA and candida are $1.66 \pm 0.01$, whereas for the pseudorandom array, it is $1.82 \pm 0.01$. These values coincide with those already obtained in [9], for the human DNA by using the more general sandbox formula. It also should be noticed that the fractal dimension of DNA coincides, up to $10^{-1}$, with the golden ratio number

$$D \cong \frac{1+\sqrt{5}}{2} \tag{35.6}$$

### 35.2.3 Representation

The (digital) representation of a DNA sequence is defined as the map of $\mathcal{S}$ into $\mathbb{R}^\ell, \ell \geq 1$. Indeed, the embedding space of representation is based on the four vectors

$$\mathbf{X}_x : \mathcal{A} \to \mathbb{R}^\ell, \quad (x \in \mathcal{A},\ \ell \geq 1)$$

in the real space $\mathbb{R}^\ell$, or almost equivalently in the complex space $\mathbb{C}^\ell$,

$$\mathbf{X}_x : \mathcal{A} \to \mathbb{C}^\ell, \quad (x \in \mathcal{A},\ \ell \geq 1)$$

so that $\mathbf{X}(x) \equiv \mathbf{X}_x$ is a $\ell$ple, which is associated with the symbol $x \in \mathcal{A}$.

The basic elements of the representation are as follows:

$$\begin{aligned}
\mathbf{X}_{\mathrm{A}} &= (x_{\mathrm{A}1}\ , x_{\mathrm{A}2}\ , \ldots , x_{\mathrm{A}\ell})\\
\mathbf{X}_{\mathrm{C}} &= (x_{\mathrm{C}1}\ , x_{\mathrm{C}2}\ , \ldots , x_{\mathrm{C}\ell})\\
\mathbf{X}_{\mathrm{G}} &= (x_{\mathrm{G}1}\ , x_{\mathrm{G}2}\ , \ldots , x_{\mathrm{G}\ell})\\
\mathbf{X}_{\mathrm{T}} &= (x_{\mathrm{T}1}\ , x_{\mathrm{T}2}\ , \ldots , x_{\mathrm{T}\ell}) \qquad (\ell \geq 1)
\end{aligned}$$

The digital representation in $\mathbb{R}^\ell$ or $\mathbb{C}^\ell$ of a $N$-length DNA sequence is the map $\mathcal{R} : \mathbb{N} \times \mathcal{A} \to \mathbb{R}^\ell$ or, for a complex representation, $\mathcal{R} : \mathbb{N} \times \mathcal{A} \to \mathbb{C}^\ell$ so that $\mathcal{S} \mapsto \mathcal{G} \in \mathbb{R}^\ell$ and for each $x_n \in \mathcal{S}$ it is

$$x_n \overset{\mathcal{R}}{\to} \mathbf{Y}(n) \quad (x_n \in \mathcal{S}\ ; \mathbf{Y}(n) \in \mathbb{R}^\ell)$$

being

$$\mathbf{Y}(n) \overset{\text{def}}{=} \mathbf{Y}(x_n) \overset{(1)}{=} \mathbf{Y}(x(n))$$

defined as follows. Each element of the DNA sequence can be considered [2] as the linear combination

$$\mathbf{Y}(n) \overset{\text{def}}{=} u_A(x_n)\mathbf{X}_{\text{A}} + u_{\text{C}}(x_n)\mathbf{X}_{\text{C}} + u_{\text{G}}(x_n)\mathbf{X}_{\text{G}} + u_{\text{T}}(x_n)\mathbf{X}_{\text{T}} \ , \ (n = 1, \dots, N) \quad (35.7)$$

The graph of $\mathbf{Y}_n = \mathbf{Y}(n)$ is $\mathcal{G}$ and if we define

$$a_n \overset{\text{def}}{=} \sum_{i=1}^{n} u_A(x_i) \ , \ c_n \overset{\text{def}}{=} \sum_{i=1}^{n} u_{\text{C}}(x_i) \ , \ g_n \overset{\text{def}}{=} \sum_{i=1}^{n} u_{\text{G}}(x_i) \ , \ t_n \overset{\text{def}}{=} \sum_{i=1}^{n} u_{\text{T}}(x_i) \quad (35.8)$$

then it is

$$a_n + c_n + g_n + t_n = n \quad (35.9)$$

so that, as a consequence of Equation (35.7) and the definition (35.8), the following identity holds:

$$n\mathbf{Y}(n) = a_n\mathbf{X}_{\text{A}} + c_n\mathbf{X}_{\text{C}} + g_n\mathbf{X}_{\text{G}} + t_n\mathbf{X}_{\text{T}}$$

We have a degeneracy (or a loop, circuit, or periodicity) if it is (see, *e.g.*, [44]) $\mathbf{Y}(n) = 0$ or, equivalently,

$$a_n\mathbf{X}_{\text{A}} + \mathbf{X}_{\text{C}} + g_n\mathbf{X}_{\text{G}} + t_n\mathbf{X}_{\text{T}} = 0 \quad (35.10)$$

If, we do not have a degeneracy, then there is a one-to-one correspondence between the DNA sequence $\mathcal{S}$ and $\mathcal{G}$, (*i.e.*, $\mathcal{S} \leftrightarrow \mathcal{G}$).

Because of the definition of the indicator function (35.2), it also can be seen that

$$u_{x_h}(x_k) = 1 \Rightarrow \mathbf{Y}(h) = \mathbf{Y}(k)$$

### 35.2.4 Representation Models

This section deals with some of the most popular representation models. In particular, we have:

- in $\mathbb{R}^1$: It is $\mathbf{X}_{\text{A}} = \mathbf{X}_{\text{C}} = \mathbf{X}_{\text{G}} = \mathbf{X}_{\text{T}} = 1$ so that the digitalization is made by the four sequences [40]: $\{u_{\text{A}}(x_n)\}$ $\{u_{\text{C}}(x_n)\}$ $\{u_{\text{G}}(x_n)\}$ $\{u_{\text{T}}(x_n)\}$
- in $\mathbb{R}^2$:
  - In [27, 44], it has been proposed that nucleotides are distributed symmetrically on the half plane $x > 0$

$$\mathbf{X}_{\text{A}} = (\sin\theta, -\cos\theta) \qquad \mathbf{X}_{\text{C}} = (\cos\theta, \sin\theta)$$

$$\mathbf{X}_{\text{G}} = (\cos\theta, -\sin\theta) \qquad \mathbf{X}_{\text{T}} = (\sin\theta, \cos\theta)$$

with $\theta \in \left(0, \dfrac{\pi}{2}\right)$, $\theta \neq \dfrac{\pi}{4}$. This method has been proposed with the aim to avoid degeneracy in the sense that the projection on the vertical axis gives four disjoint values: $(-\cos\theta, -\sin\theta, \cos\theta,$ and $\sin\theta)$. In particular, in [44], it is assumed $\theta = \pi/6$.

– In [22, 23], nucleotides are localized in the four cardinal points

$$\mathbf{X}_{\mathrm{A}} = (0\,, -1), \mathbf{X}_{\mathrm{C}} = (-1\,, 0), \mathbf{X}_{\mathrm{G}} = (1\,, 0), \mathbf{X}_{\mathrm{T}} = (0\,, 1) \tag{35.11}$$

With this choice we cannot avoid degeneracy, but the nucleotides are distributed symmetrically.

- in $\mathbb{C}^1$: The representation is given in the complex plane so that, given the imaginary unit $i$ on the vertical axis, it is

– By assuming that the Euclidean distance in $\mathbb{C}^1$ of A and C is greater than A and T, we have [7]

$$\begin{aligned}
\mathbf{X}_{\mathrm{A}} &= (1\,, i) \equiv 1 + i \equiv \sqrt{2}\left(\cos\frac{\pi}{4} + i\sin\frac{\pi}{4}\right)\\
\mathbf{X}_{\mathrm{C}} &= (-1\,, -i) \equiv -1 - i \equiv \sqrt{2}\left(\cos\frac{5\pi}{4} + i\sin\frac{5\pi}{4}\right)\\
\mathbf{X}_{\mathrm{G}} &= (-1\,, i) \equiv -1 + i \equiv \sqrt{2}\left(\cos\frac{3\pi}{4} + i\sin\frac{3\pi}{4}\right)\\
\mathbf{X}_{\mathrm{T}} &= (1\,, -i) \equiv 1 - i \equiv \sqrt{2}\left(\cos\frac{-\pi}{4} + i\sin\frac{-\pi}{4}\right)
\end{aligned}$$

– By assuming that nucleotides in $\mathbb{C}^1$ are localized on the four cardinal points [7, 15, 16]

$$\begin{aligned}
&\mathbf{X}_{\mathrm{A}} = (1\,, 0) \equiv 1, \mathbf{X}_{\mathrm{C}} = (0\,, -i) \equiv -i, \mathbf{X}_{\mathrm{G}} = (-1\,, 0) \equiv -1,\\
&\quad \mathbf{X}_{\mathrm{T}} = (0\,, i) \equiv i
\end{aligned} \tag{35.12}$$

- in $\mathbb{R}^3$: In this case, the nucleotides are located in some points of $\mathbb{R}^3$:

– in [28], it is

$$\begin{aligned}
&\mathbf{X}_{\mathrm{A}} = (1\,, 1\,, -1), \mathbf{X}_{\mathrm{C}} = (-1\,, -1\,, -1), \mathbf{X}_{\mathrm{G}} = (-1\,, 1\,, -1),\\
&\quad \mathbf{X}_{\mathrm{T}} = (1\,, -1\,, -1)
\end{aligned}$$

### 35.2.5 Constraints on the Representation in $\mathbb{R}^2$

The choice of representation is the fundamental aspect in mathematical modeling of DNA. In fact, from a statistical point of view, there might be different conclusions according to the chosen representation. In particular, if we want to avoid degeneracy, then this implies some constraints on the representation as follows:

Let

$$\mathbf{X}_{\mathrm{A}} = (x_{\mathrm{A}}\ , y_{\mathrm{A}}) \quad , \quad \mathbf{X}_{C} = (x_{\mathrm{C}}\ , y_{\mathrm{C}})$$
$$\mathbf{X}_{\mathrm{G}} = (x_{\mathrm{G}}\ , y_{\mathrm{G}}) \quad , \quad \mathbf{X}_{\mathrm{T}} = (x_{\mathrm{T}}\ , y_{\mathrm{T}})$$

be the representation of the nucleotides in $\mathbb{R}^2$, the degeneracy condition (35.10) gives

$$a_n(x_{\mathrm{A}}\ , y_{\mathrm{A}}) + c_n(x_{\mathrm{C}}\ , y_{\mathrm{C}}) + g_n(x_{\mathrm{G}}\ , y_{\mathrm{G}}) + t_n(x_{\mathrm{T}}\ , y_{\mathrm{T}}) = 0$$

that is

$$\begin{cases} a_n x_{\mathrm{A}} + c_n x_{\mathrm{C}} + g_n x_{\mathrm{G}} + t_n x_{\mathrm{T}} = 0 \\ a_n y_{\mathrm{A}} + c_n y_{\mathrm{C}} + g_n y_{\mathrm{G}} + t_n y_{\mathrm{T}} = 0 \end{cases}$$

This system, to be a nonsingular system, must admit only the trivial solution

$$a_n = c_n = g_n = t_n = 0$$

so that it must be

$$rk \begin{pmatrix} x_{\mathrm{A}} & x_{\mathrm{C}} & x_{\mathrm{G}} & x_{\mathrm{T}} \\ y_{\mathrm{A}} & y_{\mathrm{C}} & y_{\mathrm{G}} & y_{\mathrm{T}} \end{pmatrix} = 2$$

For instance, in the four cardinal points (35.11), it is

$$\begin{cases} -c_n + g_n = 0 \\ -a_n + t_n = 0 \end{cases}$$

which is solved by

$$c_n = g_n\ ,\ a_n = t_n \tag{35.13}$$

When the representation is nonsingular, for each $n$, it is $\mathbf{Y}(n) \neq 0$ so that there is a one-to-one correspondence between the location and the value of $\mathbf{Y}(n)$ as

$$\mathbf{Y}(n) = \frac{a_n}{n}\mathbf{X}_{\mathrm{A}} + \frac{c_n}{n}\mathbf{X}_{\mathrm{C}} + \frac{g_n}{n}\mathbf{X}_{\mathrm{G}} + \frac{t_n}{n}\mathbf{X}_{\mathrm{T}}$$

Even if there is a nontrivial solution of the periodicity, it should be noticed that the solution (35.13) is indeed far from being observed in real sequences in the sense that the four nucleotides are not distributed equally in the DNA. For this reason, in the following, we will focus on the cardinal representation because, according to (35.3) and (35.4), nucleotides are not distributed homogeneously.

### 35.2.6 Complex Representation

In the following, we will consider the cardinal representation (35.11) in $\mathbb{C}^1$ (35.12) so that the DNA representation is the $N$-length one-dimensional complex signal $\{Y_n\}_{n=0,\dots,N-1}$. In this case, from (35.7) and (35.12), we have

$$\begin{aligned} Y_n &= u_A(x_n) - u_C(x_n)i - u_G(x_n) + u_T(x_n)i \\ &= [u_A(x_n) - u_G(x_n)] + [u_T(x_n) - u_C(x_n)]\, i \end{aligned} \tag{35.14}$$

or

$$Y_n = \xi_n + \eta_n i \ , \ |\xi_n| + |\eta_n| = 1 \ , \ \xi_n \eta_n = 0$$

with

$$\xi_n \overset{\text{def}}{=} u_{\text{A}}(x_n) - u_{\text{G}}(x_n), \qquad \eta_n \overset{\text{def}}{=} u_{\text{T}}(x_n) - u_{\text{C}}(x_n)$$

so that the representation is a map $\mathcal{S} \to \mathbb{C}^1$, and the time series $\mathbf{Y}(n)$ is a sequence of complex numbers

$$\{Y_n\}_{n=0,\dots,N-1} \ , \qquad Y_n = \xi_n + \eta_n i$$

In the following, the complex value digital sequence $\{Y_n\}_{n=0,\dots,N-1}$, in the cardinal representation (35.12), associated with a $N$-length DNA sequence shortly will be called DNA representation (or DNA signal).

### 35.2.7 DNA Walks

DNA walk (or DNA series) is defined as the series

$$\sum Y_n, \quad n = 0, \dots, N-1 \tag{35.15}$$

which is the cumulative sum on the following DNA sequence representation:

$$\left\{ Y_0 \ , Y_0 + Y_1 \ , \dots , \sum_{s=0}^{n-1} Y_s \dots , \sum_{s=0}^{N-1} Y_s \right\}$$

Taking into account Equations (35.8) and (35.14), for the complex cardinal representation, it is

$$\begin{aligned} Z_n \overset{\text{def}}{=} \sum_{s=0}^{n-1} Y_s &= \sum_{s=0}^{n-1} \{[u_{\text{A}}(x_s) - u_{\text{G}}(x_s)] + [u_{\text{T}}(x_s) - u_{\text{C}}(x_s)]\, i\} \\ &= (a_n - g_n) + (t_n - c_n)i \end{aligned}$$

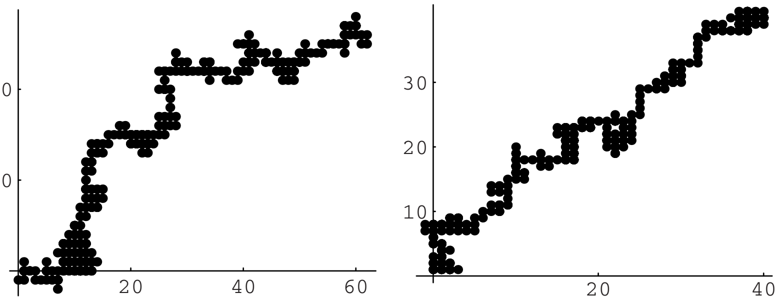


**Figure 35.4** DNA walk ($n \leq 300$) of the candida's DNA (left) and the dog's (right).

So that the DNA walk is the complex values signal $\{Z_n\}_{n=0,\ldots,N-1}$ with

$$Z_n = (a_n - g_n) + (t_n - c_n)i \tag{35.16}$$

where the coefficients $a_n, g_n, t_n$, and $c_n$ given by Equation (35.8) fulfill condition (35.9).

The DNA walk (DNA series) on a complex cardinal representation is a complex series as well. If we map the points

$$P_n = (\Re[Z_n], \Im[Z_n]) = (a_n - g_n, t_n - c_n), \ n = 0, ..., N - 1$$

whose coordinates are the real and the imaginary coefficients of each term of the DNA walk sequence, then we obtain a cluster showing the existence of some patches or some kind of self-similarity (Figures 35.4, 35.5). It should be noticed (Figure 35.5) that nearly all points of the DNA walk lie in the positive sector of the plane so that: $a_n \geq g_n$, $t_n \geq c_n$, $(n = n_0, \ldots, N)$. Both figures for the candida DNA and the dog's (Figures 35.4 and 35.5) show that a fractal behavior exists of the random sequence.

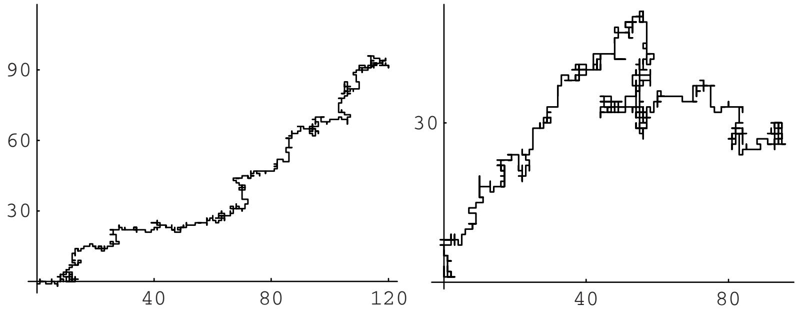


**Figure 35.5** DNA walk ($n \leq 750$) of the candida's DNA (left) and the dog's (right) for $n \leq 900$.

## 35.3 STATISTICAL CORRELATIONS IN DNA

To understand the complex organization of genomes, a central role is played by the interpretation of correlations in DNA sequence.

### 35.3.1 Long-Range Correlation

The most popular techniques for measuring correlations in a time series are:

- The direct computation of the correlation function
- Analysis of variance [35] later improved by the detrended fluctuation analysis [36]
- Power spectrum method [33, 40]
- Mutual information function [29]
- Wavelets method [3, 5, 6].

For a given sequence $\{Y_0,\ Y_1,\ \ldots,\ Y_{N-1}\}$, the variance is

$$\sigma^2 \overset{\text{def}}{=} \frac{1}{N}\sum_{i=0}^{N-1} Y_i^2 - \left(\frac{1}{N}\sum_{i=0}^{N-1} Y_i\right)^2 \tag{35.17}$$

and the variance at the distance $N-k$ is

$$\sigma_k^2 \overset{\text{def}}{=} \frac{1}{N-k}\sum_{i=0}^{N-k-1} Y_i^2 - \left(\frac{1}{N-k}\sum_{i=0}^{N-k-1} Y_i\right)^2 \tag{35.18}$$

From the variance immediately follows the standard deviation

$$\sigma = \sqrt{\sigma^2} \tag{35.19}$$

The autocorrelation at the distance $k$, $(k = 0, \ldots, N-1)$ is the sequence (see *e.g.*, [8])

$$c_k \overset{\text{def}}{=} \frac{1}{\sigma^2}\left(\frac{1}{N-k}\sum_{i=0}^{N-k-1} Y_i Y_{i+k} - \frac{1}{(N-k)^2}\sum_{i=0}^{N-k-1} Y_i \sum_{i=0}^{N-k-1} Y_{i+k}\right) \tag{35.20}$$

with $k = 0, \ldots, N-1$.

A simplified definition of correlation, in the fragment $F - N$ has been given [20] as follows:

$$c_k \overset{\text{def}}{=} \sum_{i=F}^{N-1-k} \frac{1}{N-F-k} u_{x_i}(x_{i+1+k})$$

with the indicator given by Equation (35.2).

The power spectrum can be computed as the Fourier transform of $c_k$ as follows:

$$S_k \overset{\text{def}}{=} \widehat{c_k} = \sum_{n=0}^{N-1} c_n e^{-2\pi i n k / N}$$

If $c_k = 0$, then there is no linear correlation; $c_k > 0$ means that there is a strong (linear) correlation (anticorrelation when $c_k < 0$), whereas $c_0 = 1$ doesn't give any information about correlations. A true random process has a vanishing correlation $c_h = \delta_{0h}$ and its power spectrum $S_h$ is constant. Its integral gives the Brownian motion (random walk) whose power spectrum is proportional to $1/k^2$.

It has been shown [3, 5, 45] that correlations in DNA are linear. However, the main problem of this measure is that it strongly depends on the representation and on the length of the sequence, and for nonbinary representation, it is affected by spurious results [8]. Moreover, the definition (35.20) holds only for real values of the representation.

It also has been noticed [8, 42] that to have accuracy and to avoid statistical fluctuations in the computation of the autocorrelation function, a long sequence is needed. The statistical fluctuation is $\epsilon = \dfrac{1}{\sqrt{N}}$ so that the autocorrelation is measured by $c_k \pm \dfrac{1}{\sqrt{N}}$. Therefore, when the sequence is shorter then its fluctuation is larger.

Moreover, there are several critical comments on the direct measure of correlation:

- Different sequences may exhibit the same correlation functions
- Correlation functions obtained for the whole sequence may be different for a subsequence

***35.3.1.1 Covariance Matrix.*** For a better understanding of the correlation, it also has been proposed to take into account the crosscorrelation [29] function. To define the crosscorrelation matrix, we need some basic definitions of probability; let $p_{AB}(\ell)$ be the joint probability of observing the symbol $A$ and $B$, of the alphabet $\mathcal{A} = \{\mathrm{A},\ \mathrm{C},\ \mathrm{G},\ \mathrm{T}\}$, separated by a distance $\ell$ in the DNA sequence $\mathcal{S}$, and

$$p_{A\times}(\ell) \overset{\text{def}}{=} \sum_{B=1}^{4} p_{AB}(\ell) \qquad p_{\times B}(\ell) \overset{\text{def}}{=} \sum_{A=1}^{4} p_{AB}(\ell)$$

the cumulated probability density. The cross-correlation matrix between nucleotide $A \in \mathcal{A}$ and nucleotide $B \in \mathcal{A}$ is defined as [32] follows:

$$\Gamma_{AB}(\ell) \overset{\text{def}}{=} p_{AB}(\ell) - p_{A\times}(\ell) p_{\times B}(\ell) \tag{35.21}$$

In principle, there are 16 correlation functions, but because of symmetries [29, 32], the number of independent functions reduces to nine.

Therefore, the computation of the correlation function depends on how it computes the probability on a finite $N$-length sequence.

Several methods [32] can be used to estimate the probability that enter in the Equation (35.21); the simplest is the frequency estimator. It is assumed that the probability is given as the ratio of the number of counts events $N_A$ over the total number of count $N$: $p_{AB} = \dfrac{N_{AB}}{N}$.

***35.3.1.2 Analysis of Variance and Detrended Fluctuation Analysis.*** With this method as a measure of the correlation, the variation is taken of variance along some sliding windows. Given a window of length $\sigma$, it first computes a moving average of length $\sigma$. This average can overlap or not depending on the analysis. As a second step, the variance of this moving averages sequence is computed. This variance as a function of $\sigma$ can give the information about correlation in the case of stationary data.

Detrended fluctuation analysis improves the analysis of variance, and it is based on the following steps [8, 36]:

- Sequence splitting of the $N$-length DNA series (random walk) into $\sigma = N/\ell$ nonoverlapping segments and defining the local trend in each segment
- Defining the detrended walk as the difference between the random walk and the local trend
- Computing the variance in each segment and the average of these variances over all segments.

Although this method is very popular, it has been shown [26] that it is impossible to avoid fully the influence of trends on this analysis.

***35.3.1.3 Mutual Information Function.*** This method is based on the mutual information function defined as [29, 32] follows:

$$M(\ell) \stackrel{\text{def}}{=} \sum_{AB} p_{AB}(\ell) \log_2 \frac{p_{AB}(\ell)}{p_{A\times}(\ell) p_{\times B}(\ell)}$$

which can be considered the average over all correlation functions. In absence of a correlation, it is $M(\ell) = 0$.

### 35.3.2 Power Spectrum

Let $\{Y_n\}_{n=0,\dots,N-1}$ be a given series, the discrete Fourier is the sequence

$$\widehat{Y}_s = \frac{1}{N} \sum_{n=0}^{N-1} Y_n e^{-2\pi \mathrm{i} ns/N} \quad , \quad s = 0, \dots, N-1$$

The power spectrum of the sequence $\{Y_n\}_{n=0,\dots,N-1}$, which is the mean square fluctuation, is defined as [18] follows:

$$S_k \overset{\text{def}}{=} \sum_{s=0}^{k-1} \left|\widehat{Y}_s\right|^2 \tag{35.22}$$

The power spectrum of a stationary sequence, gives an indirect measure of the autocorrelation. A long-range correlation can be detected if the fluctuations can be described by a power law so that

$$S_k \cong \alpha \frac{k}{\max\limits_{1\leq k\leq k_{\max}} [\alpha k]} \qquad 1 \leq k \leq k_{\max}$$

with $\alpha > \dfrac{1}{2}$.

The fluctuation exponent $\alpha$, with its values, characterizes a sequence as

1. Anti-correlated: $\alpha < 1/2$
2. Uncorrelated (white noise): $\alpha \cong 1/2$ Figure 35.7
3. Correlated (long-range correlated): $\alpha > 1/2$
4. $1/f$ noise: $\alpha \cong 1$
5. Nonstationary, random-walk like: $\alpha > 1$
6. Brownian noise: $\alpha \cong 3/2$

For the human DNA, [7] a long-range correlation was observed only for coding regions with $\alpha = 0.61$. However, this value also can be seen for dog's and candida's DNA (Figure 35.6) for the complete sequence (coding and noncoding regions), even if by including the noncoding regions this value is a little bit higher being $\alpha \cong 0.65$ for the dog's, and $\alpha \cong 0.62$ for the candida's DNA, respectively.

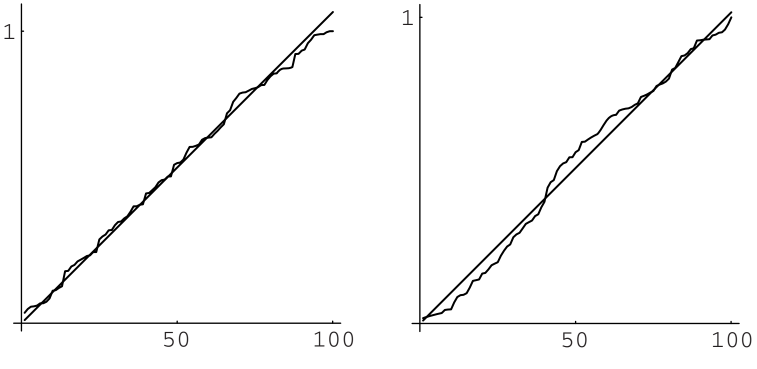


**Figure 35.6** Power spectrum for dog's DNA (left) and candida's DNA (right) with the corresponding least-square linear fit $\alpha = 0.65$ for the dog's DNA and $\alpha = 0.62$ for the candida's DNA, respectively.

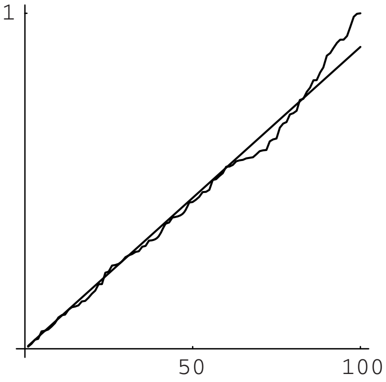


**Figure 35.7** Power spectrum for white noise (random sequence (35.23)) with the corresponding least-square linear fit: 0.5 *k*.

A pseudorandom (white noise) complex sequence similar to the cardinal complex representation (35.14) can be defined as follows:

$$R_n \overset{\text{def}}{=} (-1)^{r_1} i^{r_2} \tag{35.23}$$

with $r_1$ , $r_2$, random integer. It looks like

$$\{-1, i, 1, 1, -i, i, -1, -i, i, 1, -i, i, -1, i, 1, -i, -i, -i, -1, .....\}$$

and its fluctuation parameter is $\alpha \cong 0.5$ (see Figure 35.7).

It is interesting to note that the power spectrum for a step function

$$Y_h = \begin{cases} 1, 0 \leq h \leq (N-1)/2 \\ 0, (N-1)/2 < h \leq N-1 \end{cases}$$

is $S_k \propto 1/k^2$. This was commented [32] by the fact that for each DNA sequence, in one half of the chain the distribution of C and G is higher, whereas the distribution of A and T is lower and vice-versa in the second half. Therefore, it is expected that whatever is the representation, the power spectrum is characterized by a $1/k^2$ decay.

When the power spectrum is a power-law function

$$S(k) = S_k \propto \frac{1}{k^\alpha}$$

this function is scale invariant (like fractals), that is, $f(\lambda x) = \lambda^H f(x)$. It can be shown (see *e.g.*, [32]) that for the power-law functional dependence, in the $N \to \infty$ limit, it is

$$S(\lambda k) \propto \lambda^{1-a} S(k) \qquad S(k) = 1/k^b \tag{35.24}$$

with $k \simeq 1 - a$ and $a$ being related to the so-called Hurst exponent. In other words, for a power-law function, the power spectrum is scale invariant (like a fractal).

In particular, from Equation (35.24) it follows that when $b \simeq 1$ and $a \simeq 0$, the correlation function has a slow decay to zero, and the spectrum more properly is called $1/f$-noise (or white noise). This spectrum appears in many natural phenomena (noise in electronic devices, traffic flow, signals, radio-antenna, and turbolence).

It also has been observed [6] that for scale-invariant functions, the standard deviation (35.19), as a function of the scale $n$, is

$$\sigma(2^n) = \sigma(2^0)2^{n(H-1)} \tag{35.25}$$

where $H$ is the Hurst exponent. So that in a log–log plot

$$\log_2 \sigma(2^n) = n(H-1)\log_2 \sigma(2^0)$$

we obtain a straight line whose slope gives an estimate of $H$.

The power spectrum $1/k$ has been observed in DNA sequences [31, 33]; however, it is not yet clear what the correlation function should be (step-function, power law decay, or white-noise), and in particular if a single length scale or a multilength scale exists [32]. This scale dependence (or self-similarity) of DNA cannot yet been explained from biological point of views. A possible explanation could be the dynamic process of the evolution or maybe the functional activity inside the constrained domain (like the fractal shape of brain, lungs, *etc.*), which might have some influence on the spatial geometry [30].

The biological explanation of long-range correlations can be explained by the existence of heterogeneity in DNA (*i.e.*, a different density distribution of bases). The main questioning is about the power law spectrum: $1/k$ [39] or $1/k^2$ [32]. Indeed, it has been observed that the power spectrum is nearly flat for low and high frequencies and only the central part has a low power decay.

However, it has been observed [32, 33] that the existence of long-range correlation in DNA should be intended from a statistical point of view in the sense that far away base pairs tend to have a similar variation. In other words, this correlation should be understood as a periodic distribution of base pairs without a causality law between base pairs located at different segments far away from each other.

### 35.3.3 Complexity

The existence of repeating motifs, periodicity, and patchiness can be considered a simple behavior of sequence, whereas nonrepetitiveness or singularity is taken as a characteristic feature of complexity. To have a measure of complexity, for an $n$-length sequence, the following has been proposed [7]:

$$K = \log \Omega^{1/n}$$

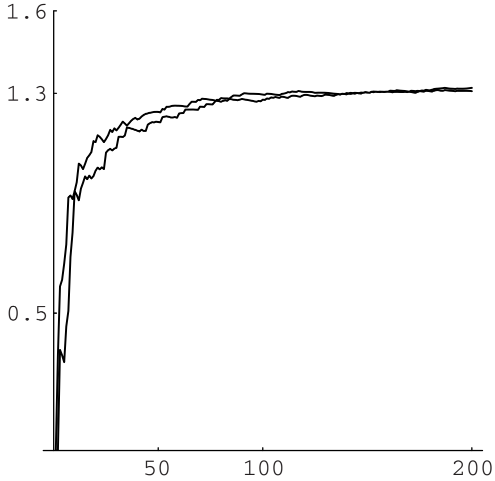


**Figure 35.8** Complexity for the first 200 base pairs of a dog's DNA and a candida's DNA.

with

$$\Omega = \frac{n!}{a_n!c_n!g_n!t_n!}$$

By using a sliding $n$-window [7] over the full DNA sequence, one can visualize the distribution of complexity on a partial fragment of the sequence. For the whole sequence, the asymptotic constant value is Figure 35.8

$$K \cong 1.3$$

## 35.4 WAVELET ANALYSIS

Wavelet analysis can be considered a good tool [3, 5, 32, 46] for studying the heterogeneity in a time series, particularly in a DNA sequence. Heterogeneity briefly can be described as follows: in some fragments of DNA, a higher concentration of nucleotides exits C and G with a poor distribution of A and T, whereas on the contrary, other fragments are more rich of A and T and poor of C and G (see Figure 35.3 and Equations (35.3) and (35.4)). Thus, a fundamental problem is to make a partition of a DNA sequence into homogenous segments. This segmentation can be done by minimizing the variance (or by maximizing the entropy [8]).

The wavelet transform expresses the signal in terms of dilated and scaled instances of the wavelet basis functions. If we call $W[f]_{x_0}$ the wavelet transform of the signal $f(x)$ computed in $x = x_0$ at the scale $2^{-n}$ and $h(x_0)$ the local Hölder exponent, then it is [3] $W[f]_{x_0} \simeq 2^{-nh(x_0)}$. Therefore, wavelet transform is one of the most expedient tools for detecting singularities. It can be used to define a generalization of the box-counting method, the so-called wavelet transform modulus maxima, to focus on the scaling behavior [3] and to visualize the multifractal property.

In this section, some fundamentals on the Haar wavelet theory will be given and applied to the analysis of DNA sequences.

### 35.4.1 Haar Wavelet Basis

The *Haar scaling function* $\varphi(x)$ is the characteristic function on [0, 1]; its family of translated and dilated scaling functions is defined as follows:

$$\begin{cases} \varphi_k^n(x) \overset{\text{def}}{=} 2^{n/2}\varphi(2^n x - k) & (0 \leq n\ ,\ 0 \leq k \leq 2^n - 1) \\ \varphi(2^n x - k) = \begin{cases} 1\ , x\ \in \Omega_k^n \\ 0\ , x\ \notin \Omega_k^n \end{cases} & \Omega_k^n \overset{\text{def}}{=} \left[\dfrac{k}{2^n}, \dfrac{k+1}{2^n}\right) \end{cases} \tag{35.26}$$

The *Haar wavelet* family $\{\psi_k^n(x)\}$ is the orthonormal basis for the following $L^2(\mathbb{R})$ functions [19]:

$$\begin{cases} \psi_k^n(x) \overset{\text{def}}{=} 2^{n/2}\psi(2^n x - k) \qquad ||\psi_k^n(x)||_{L^2} = 1 \\ \psi(2^n x - k) \overset{\text{def}}{=} \begin{cases} -1 & x \in \left[\dfrac{k}{2^n}, \dfrac{k+1/2}{2^n}\right) \\ 1 & x \in \left[\dfrac{k+1/2}{2^n}, \dfrac{k+1}{2^n}\right) \\ 0 & \text{elsewhere} \end{cases} \qquad (0 \leq n\ ,\ 0 \leq k \leq 2^n - 1) \end{cases} \tag{35.27}$$

### 35.4.2 Haar Series

Let $V_N$ be the subspace of $L^2(\mathbb{R})$ of the piecewise constant functions $y(x)$ with compact support on $\Omega_k^N$ ($N$ fixed, $k \in \mathbb{Z}$):

$$V_N \equiv \{y(x) \in L^2(\mathbb{R}) \quad y(x) = Y_k = \text{constant}$$

$$x \in \Omega_k^N \quad y(x) = 0 \quad x \notin \Omega_k^N \quad (k = 0, \ldots, N-1)\}$$

Any $y(x) \in V_N$, according to Equations (35.26) and (35.27), can have two equivalent representations [12, 13]

$$y(x) = \begin{cases} \displaystyle\sum_{k=0}^{N-1} Y_k\ \varphi_k^N(x) \\ \alpha\varphi(x) + \displaystyle\sum_{n=0}^{M-1}\sum_{k=0}^{2^n-1} \beta_k^n \psi_k^n(x) \qquad (2^M = N) \end{cases} \tag{35.28}$$

in terms of scaling functions and Haar wavelets, respectively. So the piecewise constant function $y(x)$, interpolating the points $\{x_i, Y_i\}$, with $x_i = i/(2^M - 1)$, $i = 0, \ldots, 2^M - 1$, is such that $y(x) = y(x_i) = Y_i,\ \forall\, x \in \Omega_i^N$.

In general, any function (not only piecewise constant) $F(x)$ in $L^2(\mathbb{R})$ can be reconstructed completely as Equation (35.28) in terms of wavelet series [12–14, 19]. Thus, by fixing the scale of approximation, or resolution $N = 2^M < \infty$, any function $F(x) \in L^2(\mathbb{R})$ might be approximated by its projection $\pi^N F(x)$ into $V_N$, $(N = 2^M)$, that is (in the case of Haar wavelets) by a piecewise constant function, like in Equation (35.28). Formally, both representations in Equation (35.28) are equivalent; in both cases, we still have the piecewise constant interpolation, but the wavelet representation is based on the wavelet coefficients $\alpha$, $\beta_k^n$, having some direct interpretation of the "local" behavior of the piecewise constant function (and the data as well). The coefficients can be defined as follows.

The scalar product of two functions $F(x)$, $G(x)$, of $L^2(\mathbb{R})$, is defined as

$$\langle F(x), G(x)\rangle \overset{\text{def}}{=} \int_{-\infty}^{\infty} F(x)G^*(x)\mathrm{d}x \tag{35.29}$$

It can be shown easily that Haar wavelets are orthogonal functions in the sense that

$$\begin{aligned} &\langle \varphi_k^n(x), \varphi_h^m(x)\rangle = 2^{(n+m)/2}\delta^{nm}\delta_{hk} \quad \langle \varphi(x), \psi_h^m(x)\rangle = 0 \\ &\langle \psi_k^n(x), \psi_h^m(x)\rangle = 2^{(n+m)/2}\delta^{nm}\delta_{hk} \end{aligned} \tag{35.30}$$

where $\delta_{hk}$ is the Kroenecker symbol. Because Haar wavelets are real functions, it is $\psi_k^n(x) = \psi{*}_k^n(x)$. Thus, from Equations (35.29) and (35.30), by a scalar product with the basis functions, we obtain the following wavelet coefficients:

$$\begin{aligned} \alpha_k &= \langle F(x), \varphi_k(x)\rangle = \int_{-\infty}^{\infty} F(x)\varphi_k(x)\mathrm{d}x = \int_k^{k+1} F(x)\varphi_k(x)\mathrm{d}x \\ \beta_k^n &= \langle F(x), \beta_k^n(x)\rangle = \int_{-\infty}^{\infty} F(x)\beta_k^n(x)\mathrm{d}x = \int_{k/2^n}^{(k+1)/2^n} F(x)\beta_k^n(x)\mathrm{d}x \end{aligned}$$

that is, according to Equations (35.26) and (35.27)

$$\begin{cases} \alpha_k = 2^{-M}\displaystyle\int_k^{k+1} F(x)\mathrm{d}x \\ \beta_k^n = 2^{n/2}\left[\displaystyle\int_{(k+1/2)/2^n}^{(k+1)/2^n} F(x)\mathrm{d}x - \int_{k/2^n}^{(k+1/2)/2^n} F(x)\mathrm{d}x\right] \end{cases} \tag{35.31}$$

Because the coefficients of the series (35.28) are linearly dependent on the discrete values $Y_k$, the wavelet coefficients can be computed by the (discrete) wavelet transform $\mathcal{W}_N$ as defined in the following section.

### 35.4.3 Discrete Haar Wavelet Transform

Let $\boldsymbol{Y} \equiv \{Y_i\}$, $(i = 0, \ldots, 2^M - 1,\ 2^M = N < \infty,\ M \in \mathbb{N})$ be a real and square summable time series $\boldsymbol{Y} \in \mathbb{K}^N \subset \ell^2$ (where $\mathbb{K}$ is a real field), sampled at the *dyadic points* $x_i = i/(2^M - 1)$, in the interval restricted for convenience and without restriction to $\Omega = [0, 1]$.

The *discrete Haar wavelet transform* is the $N \times N$ matrix $\mathcal{W}^N : \mathbb{K}^N \subset \ell^2 \to \mathbb{K}^N \subset \ell^2$ that maps the finite energy (column) vector $\boldsymbol{Y}$ into the finite energy (column) vector of the *wavelet coefficients* $\boldsymbol{\beta}_N = \{\alpha\ ,\ \beta^n_k\}$ as follows:

$$\begin{cases} \mathcal{W}_N \boldsymbol{Y} = \boldsymbol{\beta}_N \\ \quad \boldsymbol{\beta}_N \overset{\text{def}}{=} \{\alpha, \beta^0_0, \ldots, \beta^{M-1}_{2^{M-1}-1}\} \\ \quad \boldsymbol{Y} \overset{\text{def}}{=} \{Y_0,\ Y_1,\ \ldots,\ Y_{N-1}\} \quad (2^M = N) \end{cases} \tag{35.32}$$

Let the direct sum of matrices $A$, $B$ be defined as

$$A \oplus B = \begin{pmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{pmatrix}$$

where $\mathbf{0}$ is the matrix of zero elements. The $N \times N$ matrix $\mathcal{W}_N$ can be computed by the recursive product [12, 13]

$$\mathcal{W}_N \overset{\text{def}}{=} \left[ \prod_{k=1}^{M} \Big( (P_{2^k} \oplus I_{2^M - 2^k})(H_{2^k} \oplus I_{2^M - 2^k}) \Big) \right] \quad N = 2^M \tag{35.33}$$

of the direct sum of the following elementary matrices:

1. Identity: $I_{2^k} = \begin{pmatrix} 1 & & \mathbf{0} \\ & \underbrace{\ddots}_{2^k} & \\ \mathbf{0} & & 1 \end{pmatrix}$, which is equivalent to

$$I_2 \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad I_{2^k} \equiv \underbrace{I_2 \oplus \ldots \oplus I_2}_{k} = \begin{pmatrix} I_2 & & \mathbf{0} \\ & \underbrace{\ddots}_{k} & \\ \mathbf{0} & & I_2 \end{pmatrix}$$

2. Shuffle:

$$P_2 \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad P_4 \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad P_8 = \begin{pmatrix} 1 & & & & & & & \\ \downarrow & \rightarrow & 1 & & & & & \\ & & \downarrow & \rightarrow & 1 & & & \\ & & & & \downarrow & \rightarrow & 1 & \rightarrow \\ \rightarrow & 1 & & & & & & \\ & \downarrow & \rightarrow & 1 & & & & \\ & & & \downarrow & \rightarrow & 1 & & \\ & & & & & \downarrow & \rightarrow & 1 \end{pmatrix}$$

The arbitrary shuffle matrix $P_{2^k} = \left(a_{ij}\right);\ i, j = 1, \ldots, 2^k$ can be defined as $a_{i+1,j+2} = 1, i = 1, \ldots, 2^{k-1}, j = 1, \ldots, 2^k - 3, i = 2^{k-1}, \ldots, 2^k - 1,$ $j = 0, \ldots, 2^k - 2$, and as zero elsewhere.

3. Lattice (derived from the recursive inclusion formulas see, *e.g.*, [12, 13]):

$$H_2 \equiv \begin{pmatrix} \dfrac{1}{\sqrt{2}} & \dfrac{1}{\sqrt{2}} \\ -\dfrac{1}{\sqrt{2}} & \dfrac{1}{\sqrt{2}} \end{pmatrix} H_4 = H_2 \oplus H_2 = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1/\sqrt{2} & 1/\sqrt{2} \\ 0 & 0 & -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}, \ldots$$

and in general

$$H_{2^k} \equiv \underbrace{H_2 \oplus \ldots \oplus H_2}_{k} = \begin{pmatrix} H_2 & & \mathbf{0} \\ & \underbrace{\ddots}_{k} & \\ \mathbf{0} & & H_2 \end{pmatrix}$$

For example, with $N = 4$ and $M = 2$, assuming the empty set $I_0 \overset{\text{def}}{=} \emptyset$ as the neutral term for the direct sum $\oplus$ so that $A \oplus I_0 = I_0 \oplus A = A$, it follows from Equation (35.33)

$$\begin{aligned} \mathcal{W}_4 &= \prod_{k=1,2} \left[(P_{2^k} \oplus I_{4-2^k})(H_{2^k} \oplus I_{4-2^k})\right] \\ &= [(P_2 \oplus I_2)(H_2 \oplus I_2)]_{k=1}\, [(P_4 \oplus I_0)(H_4 \oplus I_0)]_{k=2} \end{aligned}$$

that is,

$$\mathcal{W}_4 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \tag{35.34}$$

Analogously, with $N = 8$ and $M = 3$, it is (from Equation (35.33))

$$\begin{aligned} \mathcal{W}^8 &= \prod_{k=1,2,3} \left[ (P_{2^k} \oplus I_{8-2^k})(H_{2^k} \oplus I_{8-2^k}) \right] \\ &= [(P_2 \oplus I_6)(H_2 \oplus I_6)]_{k=1} \, [(P_4 \oplus I_4)(H_4 \oplus I_4)]_{k=2} \, [(P_8 \oplus I_0)(H_8 \oplus I_0)]_{k=3} \end{aligned}$$

that is

$$\mathcal{W}^8 = \begin{pmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \tag{35.35}$$

## 35.5 HAAR WAVELET COEFFICIENTS AND STATISTICAL PARAMETERS

The wavelet coefficients $\alpha$, $\beta_k^n$ may be used to characterize easily the local variability of data. Let us define the mean value as follows.

$$\langle Y_{i,i+s} \rangle \overset{\text{def}}{=} \frac{1}{s+1} \sum_{k=i}^{i+s} Y_k$$

and, in particular, for $i = 0$, $s = N - 1$, the mean value

$$\langle \mathbf{Y} \rangle = \frac{1}{N} \sum_{k=0}^{N-1} Y_k \; ;$$

the first-order forward difference is

$$\Delta_h Y_i \equiv Y_{i+h} - Y_i \; , \quad 1 \leq h \leq 2^M - 1 - i ,$$

and the central difference is

$$\delta_h Y_i \equiv Y_{i+h} - Y_{i-h} \; , \quad 0 \leq h \leq \min(i, 2^M - 1 - 1)$$

respectively. It is

$$\delta_{h/2} Y_{i+h/2} = \Delta_h Y_i \tag{35.36}$$

and, in particular,

$$\begin{cases} \Delta_h Y_i \overset{\text{def}}{=} (Y_{i+h} - Y_i) \\ \Delta_h^2 Y_i \overset{\text{def}}{=} \Delta_h(\Delta_h Y_i) = (Y_{i+2h} - 2Y_{i+h} + Y_i) \, , \ldots \\ \Delta_h^n Y_i \overset{\text{def}}{=} (\Delta_h^{n-1} Y_{i+h} - \Delta_h^{n-1} Y_i) = \sum_{j=0}^{n} (-1)^j \binom{n}{j} Y_{i+jh+n} \end{cases}$$

the forward (finite) difference formulas, and

$$\begin{cases} \delta_h Y_i \overset{\text{def}}{=} (Y_{i+h} - Y_{i-h}) \\ \delta_h^2 Y_i \overset{\text{def}}{=} \delta_h(\delta_h Y_i) = (Y_{i+2h} - 2Y_i + Y_{i-2h}) \, , \ldots \\ \delta_h^n Y_i \overset{\text{def}}{=} (\delta_h^{n-1} Y_{i+h} - \delta_h^{n-1} Y_{i-h}) = \Delta_{2h}^n Y_{i-h/2} \end{cases}$$

the central (finite) difference formulas, respectively.

It easily can be shown (see, *e.g.*, [12, 13]) by a direct computational method that

$$\begin{cases} \beta_0^0 = 2^{-1+M/2} \Delta_{2^{M-1}} \langle Y_{0,2^{M-1}-1} \rangle \\ \beta_k^s = 2^{-1+(M-s)/2} \Delta_{2^{M-1-s}} \langle Y_{k\,2^{M-s},k\,2^{M-s}+2^{M-s-1}-1} \rangle \quad 0 < s < M - 1 \\ \beta_k^{M-1} = 2^{-1/2} \Delta_1 Y_{2k} \end{cases} \tag{35.37}$$

with $n = 0, \ldots, M-1$, $k = 0, \ldots, 2^{M-1}-1$ and

$$\Delta_{2^{M-1-n}} \langle Y_{k\,2^{M-n}, k\,2^{M-n}+2^{M-n-1}-1} \rangle = \Delta_{2^{M-1-n}} \left[ \frac{1}{2^{M-n-1}} \sum_{k\,2^{M-n}}^{k\,2^{M-n}+2^{M-n-1}-1} Y_s \right]$$

$$= \frac{1}{2^{M-n-1}} \sum_{k\,2^{M-n}}^{k\,2^{M-n}+2^{M-n-1}-1} \Delta_{2^{M-1-n}} Y_s$$

For example, with $M = 2$ and $N = 4$, we have

$$\alpha = \frac{1}{4}(Y_0 + Y_1 + Y_2 + Y_3)$$

and

$$\beta_0^0 = \frac{1}{2}\Delta_2 (Y_0 + Y_1) = \frac{1}{2}(\Delta_2 Y_0 + \Delta_2 Y_1)$$

$$= \frac{1}{2}(Y_{0+2} - Y_0 + Y_{1+2} - Y_1) = \frac{1}{2}(Y_2 - Y_0 + Y_3 - Y_1)$$

and

$$\begin{cases} \beta_0^1 = \frac{1}{\sqrt{2}}\Delta_1 Y_{2\cdot 0} = \frac{1}{\sqrt{2}}\Delta_1 Y_0 = \frac{1}{\sqrt{2}}(Y_0 - Y_1) \\ \beta_1^1 = \frac{1}{\sqrt{2}}\Delta_1 Y_{2\cdot 1} = \frac{1}{\sqrt{2}}\Delta_1 Y_2 = \frac{1}{\sqrt{2}}(Y_3 - Y_2) \end{cases}$$

When the wavelet coefficients are given, these equations can be solved with respect to the original data. With $M = 2$ and $N = 4$, we have, for example,

$$\begin{cases} Y_0 = \alpha - \frac{\beta_0^0 + \sqrt{2}\beta_0^1}{2}, Y_1 = \alpha - \frac{\beta_0^0 - \sqrt{2}\beta_0^1}{2} \\ Y_2 = \alpha + \frac{\beta_0^0 - \sqrt{2}\beta_1^1}{2}, Y_3 = \alpha + \frac{\beta_0^0 + \sqrt{2}\beta_1^1}{2} \end{cases}$$

According to Equation (35.37), the first wavelet coefficient $\alpha$ represents the average value of the sequence, and the other coefficients $\beta$ represent the finite differences. The wavelet coefficients $\beta$s, also called details coefficients, are connected strictly with the first-order properties of the discrete time series.

Concerning the variance, from definition (35.17), we obtain by a direct computation its expression in terms of wavelet coefficients [43]:

$$\sigma^2 = \frac{1}{N}\sum_{n=0}^{M-1}\sum_{k=0}^{2^n-1}\left(\beta_k^n\right)^2 \qquad (N = 2^M) \tag{35.38}$$

The Hurst exponent, in terms of wavelet coefficients, can be evaluated by the following

**Theorem 35.1** *The Hurst exponent is given by*

$$H = \frac{1}{2} + \log_2 \sqrt[n]{\frac{\left[\sum_{k=0}^{2^n-1}(\beta_k^n)^2\right]^{1/2}}{|\beta_0^0|}} \tag{35.39}$$

*Proof:* Taking into account Equations (35.19) and (35.38), definition (35.25) becomes the following:

$$\frac{1}{2^{n/2}}\left[\sum_{k=0}^{2^n-1}(\beta_k^n)^2\right]^{1/2} = |\beta_0^0|2^{n(H-1)}$$

$$\log_2\left\{\frac{1}{2^{n/2}}\left[\sum_{k=0}^{2^n-1}(\beta_k^n)^2\right]^{1/2}\right\} - \log_2|\beta_0^0| = n(H-1)$$

that is,

$$\frac{1}{n}\log_2\frac{1}{2^{n/2}}\frac{\left[\sum_{k=0}^{2^n-1}(\beta_k^n)^2\right]^{1/2}}{|\beta_0^0|} = (H-1)$$

$$\log_2 2 + \frac{1}{n}\log_2\frac{1}{2^{n/2}}\frac{\left[\sum_{k=0}^{2^n-1}(\beta_k^n)^2\right]^{1/2}}{|\beta_0^0|} = H$$

from which Equation (35.39) follows. ■

## 35.6 ALGORITHM OF THE SHORT HAAR DISCRETE WAVELET TRANSFORM

To reduce the computational complexity of the wavelet transform shown in Equations (35.32) and (35.33), the sequence **Y** can be sliced into subsequences, and the wavelet transform is applied to each slice. In fact, the wavelet transform (35.32) and (35.33) implies the computation of $N = 2^M$ wavelet coefficients at the resolution $M$ with $N$ basis functions $\psi_k^n(t)$ involved and a computation complexity $\mathcal{O}(N^2)$. However, if we consider only $p = 2^m \leq N$ basis functions, then the complexity reduces to $\mathcal{O}(pN)$. This can be obtained by simply slicing the data with a fixed window, as it usually is done, for instance, in the local sine and cosine transforms or in the wavelet packet decomposition. With the reduced Haar transform, it is possible both to reduce the number of basis functions and the computational complexity and to keep unchanged the piecewise constant interpolation in Equation (35.28).

**Algorithm 35.1** *Let the set $\boldsymbol{Y} = \{Y_i\}_{i=0,\dots,N-1}$ of $N$ data be, segmented into $\sigma = N/p$, $(1 \leq \sigma \leq N)$ segments of $p = 2^m$ data:*

$$\boldsymbol{Y} = \{Y_i\}_{i=0,\dots,N-1} = \bigoplus_{s=0}^{\sigma-1} \boldsymbol{Y}^s , \qquad \boldsymbol{Y}^s \equiv \{Y_{sp},\ Y_{sp+1}, \dots,\ Y_{sp+p-1}\}$$

$$(s = 0, \dots, N-p;\ 1 \leq p \leq N)$$

*the p-parameters short (reduced or windowed) discrete Haar wavelet transform $\mathcal{W}^{p,\sigma}\boldsymbol{Y}$ of $\boldsymbol{Y}$ is defined as follows:*

$$\begin{cases} \mathcal{W}^{p,\sigma} \equiv \bigoplus_{s=0}^{\sigma-1} \mathcal{W}^p \ , \ \boldsymbol{Y} = \bigoplus_{s=0}^{\sigma-1} \boldsymbol{Y}^s \\ \mathcal{W}^{p,\sigma}\boldsymbol{Y} = \left(\bigoplus_{s=0}^{\sigma-1} \mathcal{W}^p\right) \boldsymbol{Y} = \left(\bigoplus_{s=0}^{\sigma-1} \mathcal{W}^p \boldsymbol{Y}^s\right) \\ \mathcal{W}^{2^m}\boldsymbol{Y}^s = \left\{\alpha_0^{0(s)},\ \beta_0^{0(s)},\ \beta_0^{1(s)},\ \beta_1^{1(s)}, \dots,\ \beta_{2^{m-1}-1}^{m-1(s)}\right\} \ (2^m = p) \end{cases}$$

In general, for the vector of $2^M$ elements, $\boldsymbol{Y} = \{Y_i\}_{i=0,\dots,2^M-1}$, the Haar wavelet transform is the vector $\mathcal{W}^{2^M}\boldsymbol{Y}$, whereas there are different reduced transforms that can be done with one of the following matrices $\{\mathcal{W}^{2^i,2^j}\}_{i+j=M}$. To transform the vector $\boldsymbol{Y}$, instead of using the $N \times N$ matrix $\mathcal{W}^N$ up to the resolution $M = \log_2 N$, we can use the $N \times N$ matrix $\mathcal{W}^{p,\sigma}$, which is the direct sum of $p \times p$ matrices: $\mathcal{W}^{p,\sigma} = \bigoplus_{s=0}^{\sigma-1} \mathcal{W}^p$, so that in each segment $\boldsymbol{Y}^s$, we will have the resolution $m = \log_2 p$.

For example, the reduced wavelet transform $\mathcal{W}^{4,2}$, (to be compared with $\mathcal{W}^8$ of Equation (35.35)), expressed as follows:

$$\mathcal{W}^{4,2} = \mathcal{W}^4 \oplus \mathcal{W}^4$$

that is

$$\mathcal{W}^{4,2} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \tag{35.40}$$

So that for the given vector $\boldsymbol{Y} = \{Y_0, Y_1, \ldots, Y_7\}$, we might have two distinct representations:

- A single vector of eight components: $\mathcal{W}^8\boldsymbol{Y}$ with $\mathcal{W}^8$ given by Equation (35.33) as a result of the discrete Haar wavelet transform
- Two four-component vectors $\mathcal{W}^{4,2}\boldsymbol{Y} = \{\mathcal{W}^4\boldsymbol{Y}^0, \mathcal{W}^4\boldsymbol{Y}^1\}$ with $\boldsymbol{Y}^0 = \{Y_0, \ldots, Y_3\}$, $\boldsymbol{Y}^1 = \{Y_4, \ldots, Y_7\}$ as a result of the reduced discrete Haar wavelet transform $\mathcal{W}^{4,2}$ of Equation (35.40).

The reduced transform $\mathcal{W}^{4,2}$ gives some advantages in the matrix representations; it maps the data $\boldsymbol{Y}$ into a cluster of points in the space of the wavelet coefficients (*i.e.*, the row of two vectors $\{\mathcal{W}^4\boldsymbol{Y}^0, \ \mathcal{W}^4\boldsymbol{Y}^1\}$).

In general, for the vector of $2^M$ elements, $\boldsymbol{Y} = \{Y_i\}_{i=0,\ldots,2^M-1}$, the Haar wavelet transform is the vector $\mathcal{W}^{2^M}\boldsymbol{Y}$, whereas the reduced analysis can be done with one of the following matrices $\{\mathcal{W}^{2^i,2^j}\}_{i+j=M}$—of course, when $\sigma = 1 \rightarrow p = N$ and $\mathcal{W}^{p\sigma}\boldsymbol{Y} = \mathcal{W}^N\boldsymbol{Y}$.

The short wavelet transform at the resolution $m = \log_2 p$, involves only $p$ wavelet basis functions. Compared with the wavelet transform on the sequence of data, this implies a faster computation for the wavelet coefficients.

## 35.7 CLUSTERS OF WAVELET COEFFICIENTS

Significant information on a time series can be derived not only from the wavelet coefficients but also from clusters of wavelet coefficients.

For the $N = 2^M$ length real vector $\boldsymbol{Y}$, the wavelet transform $\mathcal{W}^N\boldsymbol{Y}$ represents a point in the $N$-dimensional Euclidean space

$$\mathbb{R}^N : \left(\alpha,\ \beta^0_0,\ \beta^1_0,\ \ldots,\ \beta^{M-1}_{2^{M-1}-1}\right)$$

of the wavelet coefficients.

For the $N = 2^M$ length complex vector $\boldsymbol{Y}$, the wavelet transform is applied to the real $\mathcal{W}^N\Re(\boldsymbol{Y})$ and to the imaginary part $\mathcal{W}^N\Im(\boldsymbol{Y})$ and gives either one point in

$$\mathbb{C}^N = \mathbb{R}^{2N} : \left(\alpha,\ \beta^0_0,\ \beta^1_0,\ldots,\beta^{M-1}_{2^{M-1}-1},\ \alpha^*,\ \beta^{*0}_{\ \ 0},\ \beta^{*1}_{\ \ 0},\ldots,\beta^{*M-1}_{\ \ 2^{M-1}-1}\right)$$

or two points in

$$\mathbb{R}^N \times \mathbb{R}^N : \left(\alpha,\ \beta^0_0,\ \beta^1_0,\ldots,\beta^{M-1}_{2^{M-1}-1}\right) \times \left(\alpha^*,\ \beta^{*0}_{\ \ 0},\ \beta^{*1}_{\ \ 0},\ldots,\beta^{*M-1}_{\ \ 2^{M-1}-1}\right)$$

or a cluster of $N$ points in the product of two-dimensional spaces

$$\prod_{i=1}^{N}\mathbb{R}^2_i : \left(\alpha,\ \alpha^*\right) \times \left(\beta^0_0,\ \beta^{*0}_{\ \ 0}\right) \times \left(\beta^1_0,\ \beta^{*1}_{\ \ 0}\right) \times \ldots \times \left(\beta^{M-1}_{2^{M-1}-1},\ \beta^{*M-1}_{\ \ 2^{M-1}-1}\right)$$

where the stars denote the wavelet coefficients of $\Im(\boldsymbol{Y})$. In each two-dimensional (phase) space $\mathbb{R}^2_i$, there is only one point, and these single points do not give any significant information about the existence of some autocorrelation of data. By using, instead, the $p$-parameter short Haar wavelet transform, we can analyze the cluster of points

$$\left(\mathcal{W}^p\Re(\boldsymbol{Y}^s), \mathcal{W}^p\Im(\boldsymbol{Y}^s)\right)\ s = 0,\ldots,\sigma = N/p$$

in the $2p$-dimensional space $\mathbb{R}^p \times \mathbb{R}^p$, that is,

$$\left(\alpha\ ,\alpha^*\right)\ ,\left(\beta^0_0\ ,\beta^{*0}_{\ \ 0}\right),\ldots\left(\beta^{p-1}_{2^{p-1}-1}\ ,\beta^{*p-1}_{\ \ 2^{p-1}-1}\right)$$

For a complex sequence $\{Y_k\}_{k=0,\ldots,N-1} = \{x_k + i\ y_k\}_{k=0,\ldots,N-1}$, we can consider the correlations (if any) between the wavelet coefficients of the real part $\{x_k\}_{k=0,\ldots,N-1}$ against the imaginary coefficients $\{y_k\}_{k=0,\ldots,N-1}$. This can be realized by the cluster algorithm of Table 35.1.

This algorithm enables us to construct clusters of wavelet coefficients and to study the correlation between the real and imaginary coefficients of the DNA representation and DNA walk, as given in the following section.

**Table 35.1**

| | |
|---|---|
| $\left\{Y_0, Y_1, \dots Y_p\right\} \oplus \left\{Y_{p+1}, Y_{p+2}, \dots Y_{2p}\right\} \oplus .....$ | Complex values sequence |
| $\Downarrow$ | |
| $x_i = \Re(Y_i)\ ,\ y_i = \Im(Y_i)$ | Real values sequences |
| $\Downarrow$ | |
| $\left\{x_0, x_1, \dots x_p\right\} \oplus \left\{x_{p+1}, x_{p+2}, \dots x_{2p}\right\} \oplus .....$<br>$\left\{y_0, y_1, \dots y_p\right\} \oplus \left\{y_{p+1}, y_{p+2}, \dots y_{2p}\right\} \oplus .....$ | Real sequences |
| $\Downarrow$ | |
| $\mathcal{W}^p \left\{x_0, x_1, \dots x_p\right\} \oplus \mathcal{W}^p \left\{x_{p+1}, x_{p+2}, \dots x_{2p}\right\} \oplus .....$<br>$\mathcal{W}^p \left\{y_0, y_1, \dots y_p\right\} \oplus \mathcal{W}^p \left\{y_{p+1}, y_{p+2}, \dots y_{2p}\right\} \oplus .....$ | Wavelet transform |
| $\Downarrow$ | |
| $\left\{\alpha, \beta_0^0, \beta_0^1, \beta_1^1, \dots\right\}_1 \oplus \left\{\alpha, \beta_0^0, \beta_0^1, \beta_1^1, \dots\right\}_2 \oplus .....$<br>$\left\{\alpha^*, \beta^{*0}_{\ 0}, \beta^{*1}_{\ 0}, \beta^{*1}_{\ 1}, \dots\right\}_1 \oplus \left\{\alpha^*, \beta^{*0}_{\ 0}, \beta^{*1}_{\ 0}, \beta^{*1}_{\ 1}, \dots\right\}_2 \oplus .....$ | Wavelet coefficients |
| $\Downarrow$ | |
| $\{(\alpha, \alpha^*)\}_1 \oplus \{(\alpha, \alpha^*)\}_2 \oplus \{(\alpha, \alpha^*)\}_3\ .....$<br>$\left\{\left(\beta_0^0, \beta^{*0}_{\ 0}\right)\right\}_1 \oplus \left\{\left(\beta_0^0, \beta^{*0}_{\ 0}\right)\right\}_2 \oplus \left\{\left(\beta_0^0, \beta^{*0}_{\ 0}\right)\right\}_3 .....$<br>$\left\{\left(\beta_0^1, \beta^{*1}_{\ 0}\right)\right\}_1 \oplus \left\{\left(\beta_0^1, \beta^{*1}_{\ 0}\right)\right\}_2 \oplus \left\{\left(\beta_0^1, \beta^{*1}_{\ 0}\right)\right\}_3 .....$<br>$\vdots \quad \vdots \quad \vdots$ | Clusters |

### 35.7.1 Cluster Analysis of the Wavelet Coefficients of the Complex DNA Representation

Given a DNA sequence, we can start by plotting the wavelet transform of the correlation matrix (35.2), which is the matrix of the Haar wavelet transform applied to each row of $u_{hk}$. However, because there are only finite jumps (Figure 35.9, left), the resulting picture does not give any interesting perspective in the comprehension of DNA (Figure 35.10) better than Figure 35.1

The cluster algorithm of Table 35.1, instead, when applied to the complex representation sequence (35.14), which is in the form

$$\{-1, i, 1, 1, -i, i, -1, -i, i, 1, -i, i, -1, i, 1, -i, -i, -i, -1, .....\}$$

shows that the values of the wavelet coefficients belong to some discrete finite sets (Figure 35.11). If we compare the clusters of Figure 35.11 with the clusters (Figure 35.12) of the pseudorandom sequence (35.23), which is similar to the previous sequence, then we can see that the set of wavelet coefficients is larger (still discrete) than the set for the DNA (Equation 35.11) although the detail coefficients have (more or less) the same values (see also Figure 35.9).

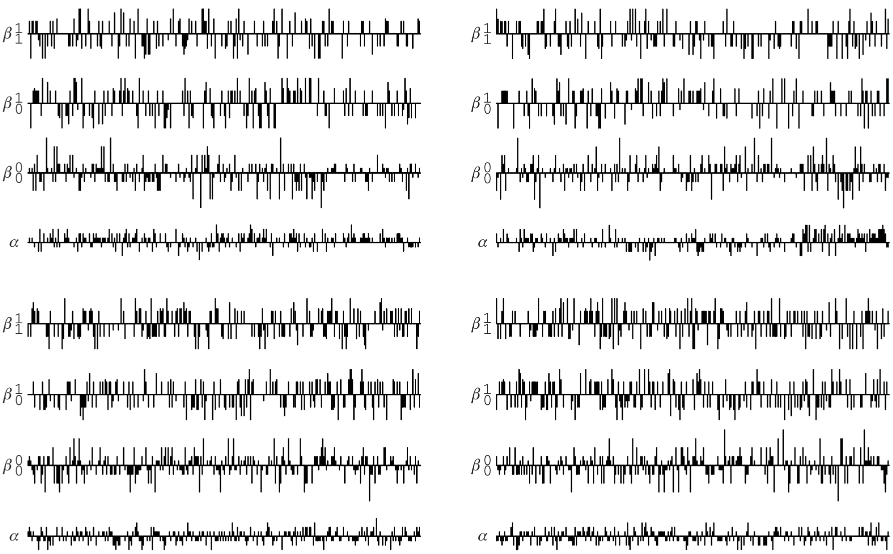


**Figure 35.9** Wavelet coefficients for the real (left column) and imaginary part of the complex representation in the fourth short Haar wavelet transform of a dog's DNA (top) compared with the pseudorandom sequence (35.23), ($n \leq 1200$).

***35.7.1.1 Detection of Misplaced Base Pairs.*** With the wavelet clustering algorithm (Table 35.1) it is easy to identify the existence of some misplaced base pairs. In fact, this pathological distribution of nucleotides implies some broken symmetries in the cluster. To show this, let us take the first 1200 terms of the sequence and make

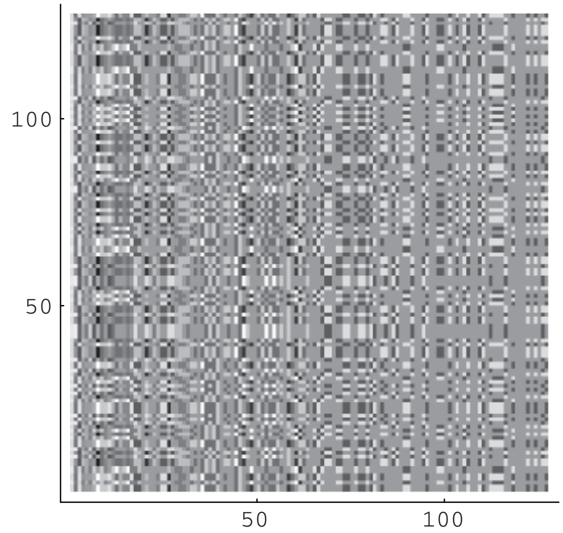


**Figure 35.10** Haar wavelet transform of the correlation matrix ($n \leq 128$) of the dog's DNA.

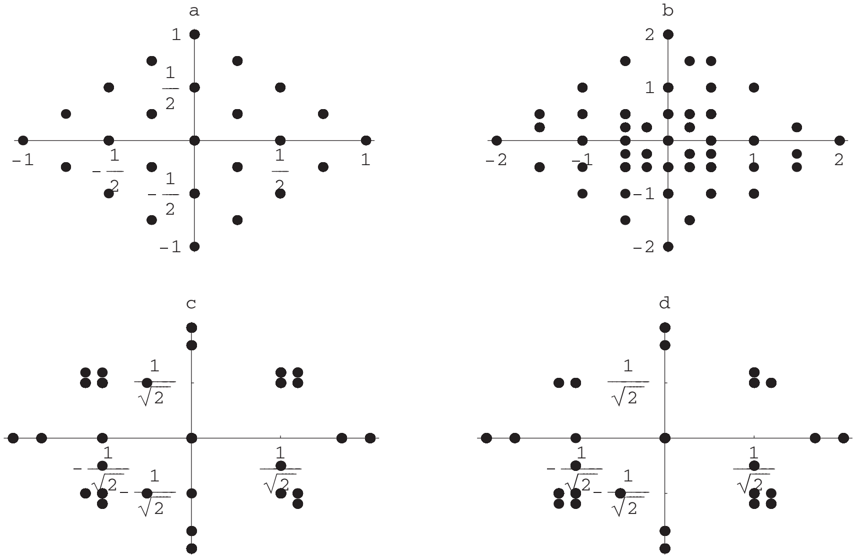


**Figure 35.11** Cluster analysis of the fourth short Haar wavelet transform of the complex representation series ($n \leq 1200$) of the dog's DNA: a) $(\alpha, \alpha^*)$ ; b) $(\beta_0^0, \beta^{*0}_0)$ ; c) $(\beta_0^1, \beta^{*1}_0)$ ; d) $(\beta_1^1, \beta^{*1}_1)$.

these changes:

$$Y_i \Rightarrow Y_i = 0 \qquad i = 100, 200, 300, 400, \ldots, 1100, 1200$$

These anomalous terms, which are very little compared with the length of the sequence (being only 1%), can be detected easily by the cluster analysis because some

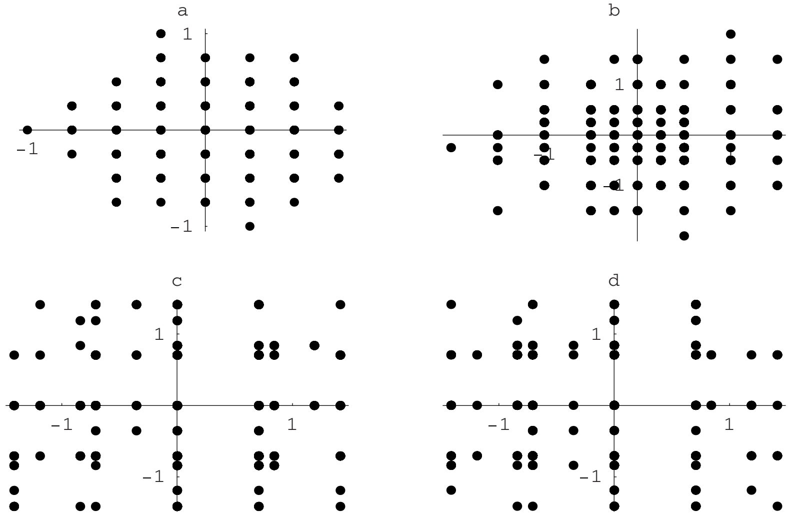


**Figure 35.12** Cluster analysis of the fourth short Haar wavelet transform of the pseudorandom sequence (35.23) ($n \leq 1200$): a) $(\alpha, \alpha^*)$; b) $(\beta_0^0, \beta^{*0}_0)$; c) $(\beta_0^1, \beta^{*1}_0)$; d) $(\beta_1^1, \beta^{*1}_1)$.

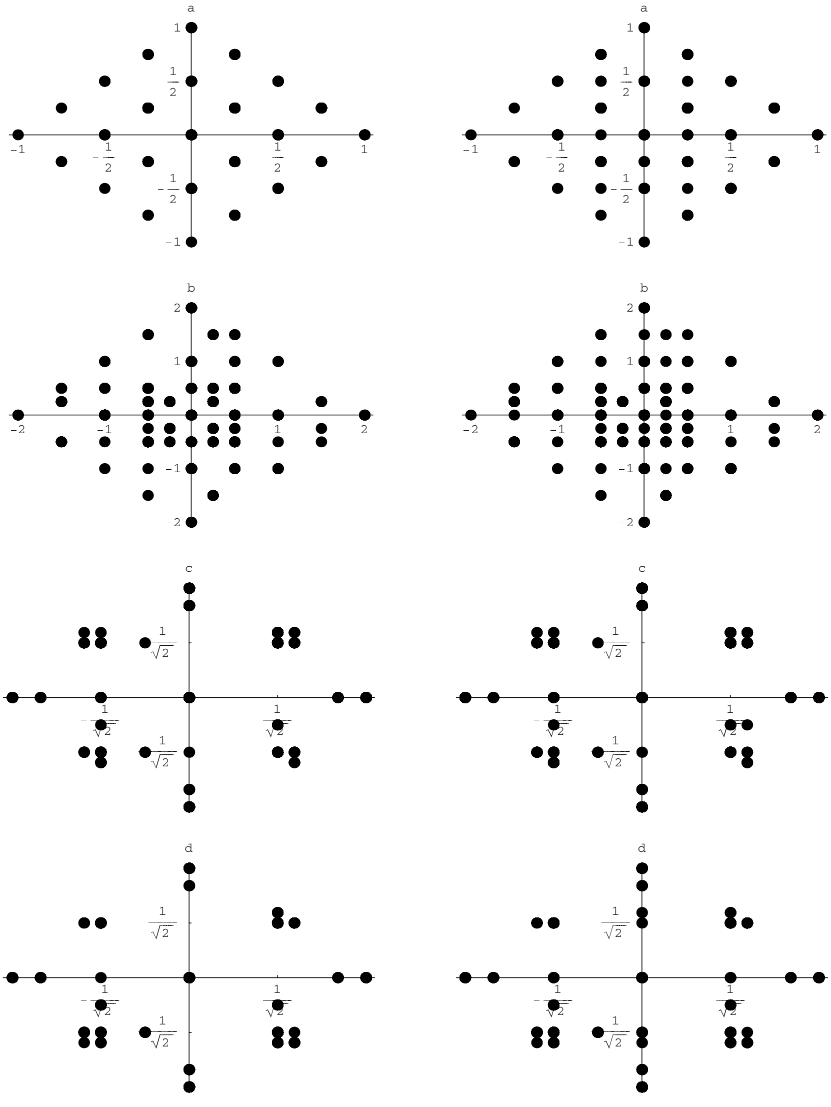


**Figure 35.13** Cluster analysis of the fourth short Haar wavelet transform of the complex representation series ($n \leq 1200$) of the dog's DNA without (left) and with some misplaced terms (right): a) $(\alpha, \alpha^*)$; b) $(\beta_0^0, \beta^*{}_0^0)$; c) $(\beta_0^1, \beta^*{}_0^1)$; d) $(\beta_1^1, \beta^*{}_1^1)$.

new spots appear (Figure 35.13 right column a, b). The anomalies can be seen by comparing the clusters of the first two rows (a and b).

As a second example, let us consider the following:

$$Y_i \Rightarrow Y_i = i \qquad i = 100, 200, 300, 400, \ldots, 1100, 1200$$

where there is very little difference from the original sequence. Although, in the first example, the digit 0 does not belong to the representation values $\{1, \ -1, \ i, \ -i\}$, in this case, $i$ belongs to them. Nevertheless, this difference can be detected by the clustering analysis. It is shown (Figure 35.14) that the clusters coincide with the original signal except for the coefficients $\left(\beta_1^1, \beta^{*1}_1\right)$ (Figure 35.14, bottom row, d).

As a third example, let us consider the case in which only two terms have been changed

$$Y_{400} = -i \Rightarrow i, \quad Y_{401} = 1 \Rightarrow i$$

so that there are only 0.1% of changes. Also, in this case, this anomaly can be detected easily by the clusters of wavelet coefficients $\left(\beta_0^0, \beta^{*0}_0\right)$, (Figure 35.15).

### 35.7.2 Cluster Analysis of the Wavelet Coefficients of DNA Walks

Let us now consider DNA walks on cardinal complex representation and the corresponding wavelet analysis.

As shown in Figures 35.4 and 35.5, the DNA walks seem to show a self-similar structure. Let us first compute the global fractal estimate with wavelet coefficients as [43]

$$\frac{\langle(\beta_0^0)^2 + (\beta^{*0}_0)^2\rangle_s}{\langle(\beta_0^1)^2 + (\beta^{*1}_0)^2 + (\beta_1^1)^2 + (\beta^{*1}_1)^2\rangle_s} = 1.6 \quad (s = 0, \ldots, (N-1)/4)$$

Moreover, by using the power spectrum method of Equation (35.22), the fluctuation exponent of the dog's DNA walk coincides with the same coefficient computed for the cardinal sequence (see Fig. 35.16) being $\alpha = 0.65$.

For each complex DNA walk, two sets of wavelet coefficients correspond to the real and complex coefficients of the complex values of Equations (35.15) and (35.16). However, even if the real and complex coefficients of the DNA walk show some nonlinear patterns (Figures 35.4 and 35.5), the detail coefficients range in some discrete sets of values. It is shown by a direct computation that the jumps from one value to another belong to some discrete sets (see *e.g.*, Figures 35.17 and 35.18).

In other words, the real and imaginary coefficients of the DNA walk increase with a given law, and the distribution of the nucleotides must follow this rule. Moreover, it should be noticed that, except for the coefficients $(\alpha, \alpha^*)$, the other wavelet coefficients are distributed on symmetric grids (Figures 35.17 and 35.18), and the patterns of the grids look similar to the grids obtained for the DNA cardinal complex representation.

It should be noted that the presence of correlations in the dog's DNA does not depend on the simple (random-like) structure of the complex representation. In the sense that even if the DNA representation looks like the pseudorandom sequence (35.23), the wavelet (detail) coefficients are quantized and symmetrically distributed such that the detail coefficients of both the representation and the DNA walks have discrete finite values (see Figures 35.11, 35.17, 35.18), in particular,

$$\left|\beta_0^0 \pm \beta^{*0}_0\right| \leq 2$$

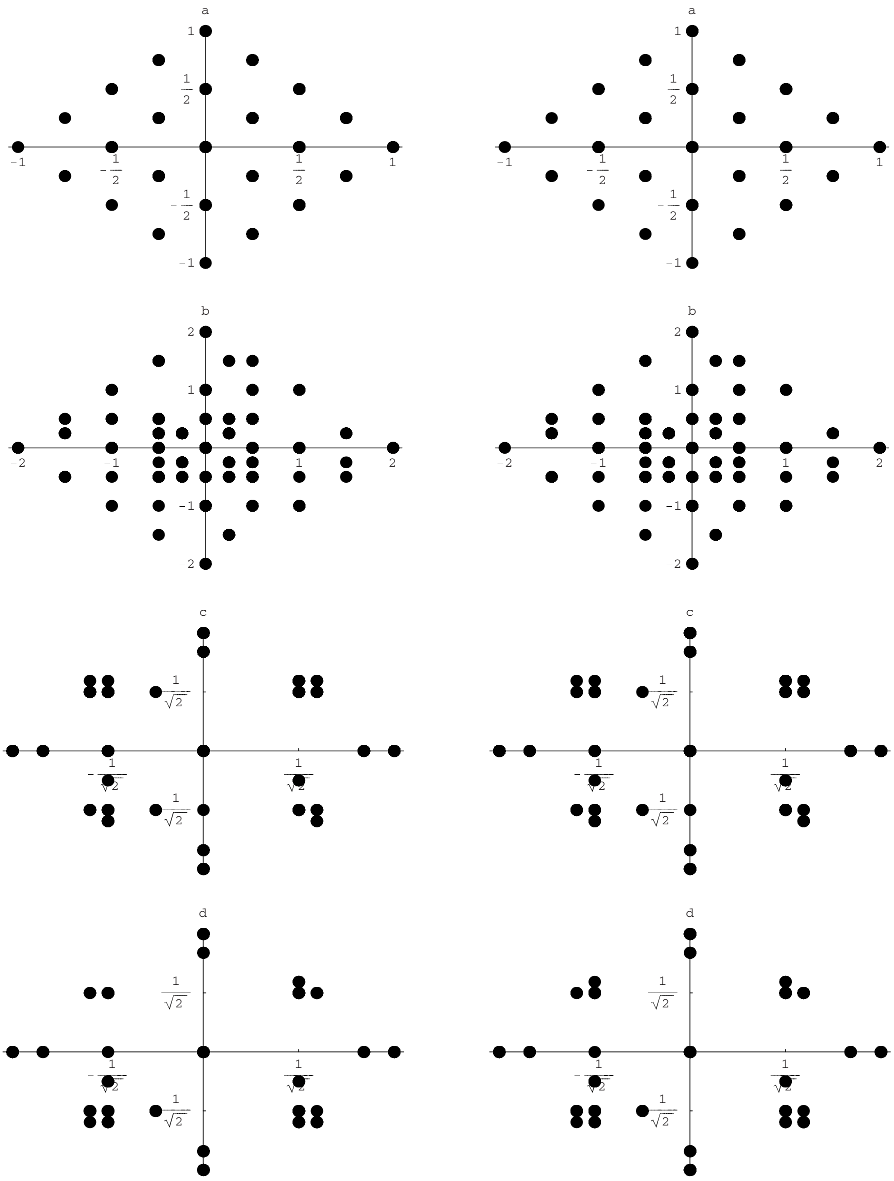


**Figure 35.14** Cluster analysis of the fourth short Haar wavelet transform of the complex representation series ($n \leq 1200$) of the dog's DNA without (left) and with some misplaced terms (right): a) $(\alpha, \alpha^*)$; b) $\left(\beta^0_0, \beta^{*0}_0\right)$; c) $\left(\beta^1_0, \beta^{*1}_0\right)$; d) $\left(\beta^1_1, \beta^{*1}_1\right)$.

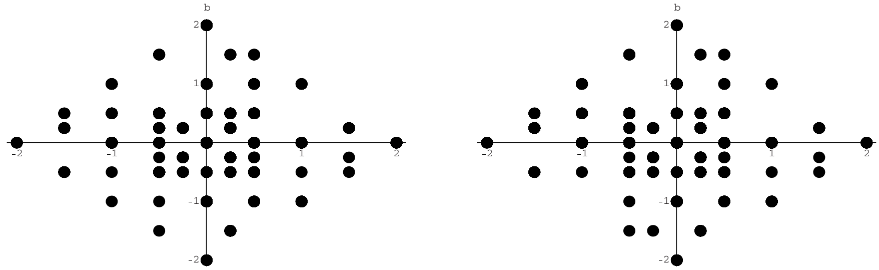


**Figure 35.15** Cluster analysis of the fourth short Haar wavelet transform of the complex representation series ($n \leq 1200$) of the dog's DNA without (left) and with some misplaced terms (right): $\left(\beta_0^0, \beta^*{}_0^0\right)$.

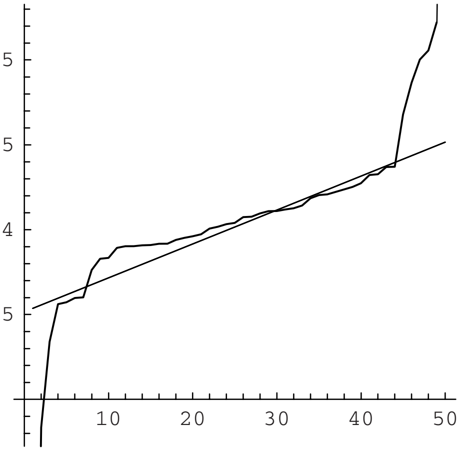


**Figure 35.16** Power spectrum for dog's DNA walk with the corresponding least-squares linear fit $\alpha = 0.65$.

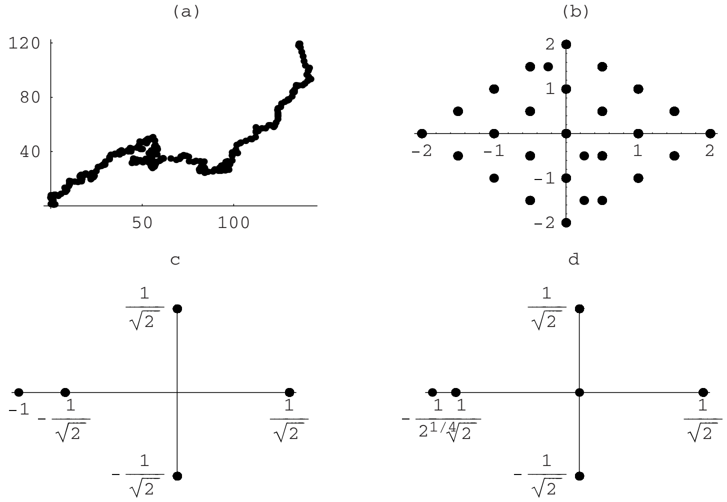


**Figure 35.17** Cluster analysis of the fourth short Haar wavelet transform of the DNA walk ($n \leq 1200$) of the dog in the planes: a) $(\alpha, \alpha^*)$; b) $\left(\beta_0^0, \beta^*{}_0^0\right)$; c) $\left(\beta_0^1, \beta^*{}_0^1\right)$; d) $\left(\beta_1^1, \beta^*{}_1^1\right)$.

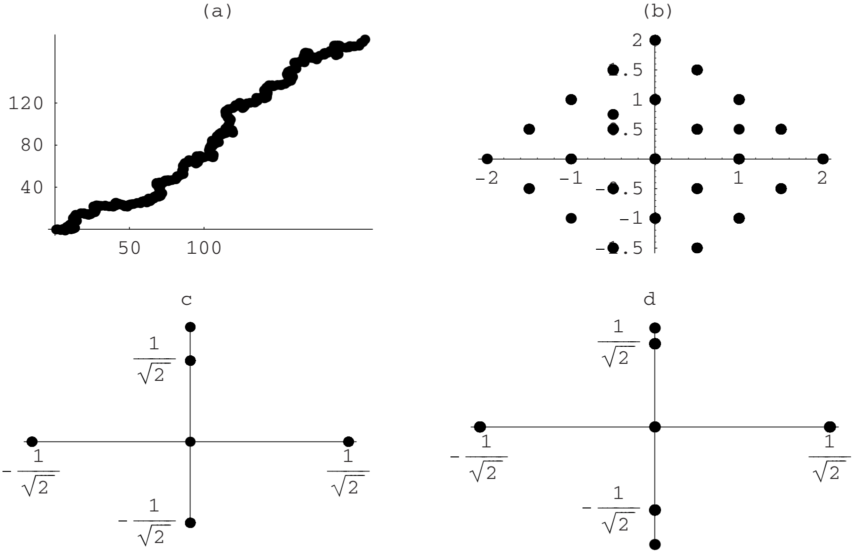


**Figure 35.18** Cluster analysis of the fourth short Haar wavelet transform of the DNA walk ($n \leq 1200$) of the candida in the planes: a) $(\alpha, \alpha^*)$; b) $(\beta^0_0, \beta^{*0}_0)$; c) $(\beta^1_0, \beta^{*1}_0)$; d) $(\beta^1_1, \beta^{*1}_1)$.

This is not true for the pseudorandom sequence, because the wavelet coefficients of the sequence still are quantized (see Figure 35.12), whereas the wavelet coefficients of the corresponding random walk are distributed randomly in the phase plane (Figure 35.19).

Also, the wavelet coefficients of the fourth short Haar wavelet transform of the DNA walk give more information than the wavelet analysis of the DNA sequence.

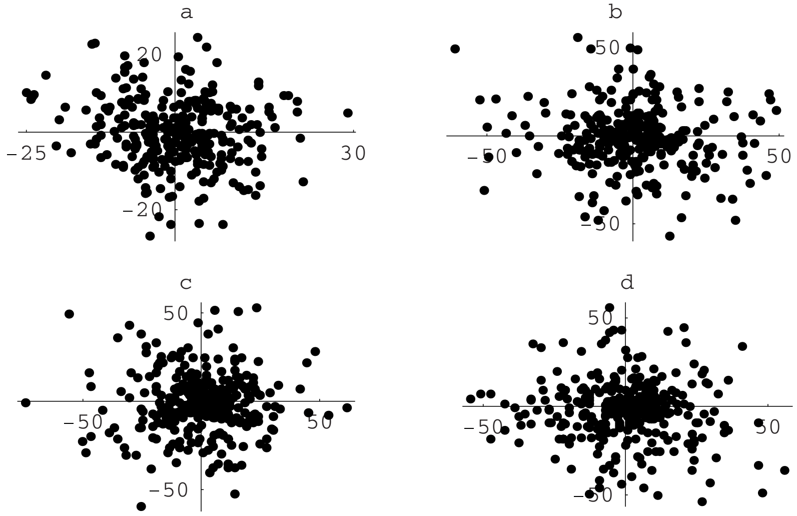


**Figure 35.19** Cluster analysis of the fourth short Haar wavelet transform of the random walk ($n \leq 1200$) of a random sequence in the planes: a) $(\alpha, \alpha^*)$; b) $(\beta^0_0, \beta^{*0}_0)$; c) $(\beta^1_0, \beta^{*1}_0)$; d) $(\beta^1_1, \beta^{*1}_1)$.

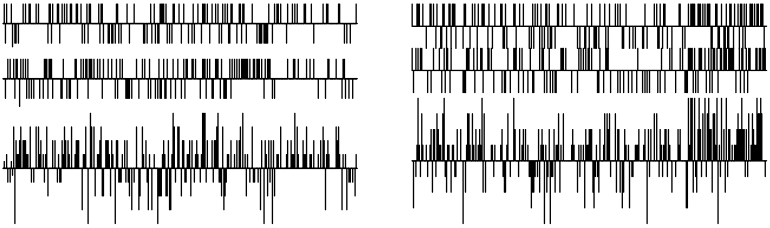


**Figure 35.20** Wavelet (detail) coefficients for the real (left column) and the imaginary part of the complex representation in the fourth short Haar wavelet transform of the dog's DNA walk ($n \leq 1200$).

It is shown that the wavelet coefficients at the higher frequencies are bounded (Figure 35.20), whereas $\beta_0^0$ shows some periodic oscillations.

## 35.8 CONCLUSION

In this chapter, the definition of a complex indicator function has been given for the DNA sequences. The indicator, applied to the dog's DNA and to the candida's DNA, has provided a pair of complex strings analyzed with the wavelet transform. By using the wavelet transform together with some algorithms (short Haar transform, and clusters of wavelet coefficients), it has been shown that the random walks have a fractal behavior, with respect to the scaling coefficient $\alpha$ and some (unexpected) symmetries and quantization for the remaining wavelet coefficients $\left(\beta_0^0, \beta_0^1, \beta_1^1\right)$ both for the real coefficients and the imaginary coefficients of the (complex) random walk. A very interesting correlation has been shown in the comparison of the random walk with its rate of change.

It should be noted that when comparing the clusters of dog and candida some slight changes in the clusters appear (but not in the quantized values of the wavelet coefficients). This could offer the possibility to organize a different classification of DNA sequences.

It has been shown that for DNA sequences, the following properties hold: heterogeneous distribution of nucleotides, long-range correlated, self-similarity, and symmetries. In particular,

1. The correlation matrix has been defined that enables us to have a simple two-dimensional representation of DNA. It has been shown that the resulting picture (see, *e.g.*, Figure 35.2, 35.1) is a fractal with the fractal dimension shown in Equations (35.5) and (35.6) coinciding with the golden ratio $\dfrac{1+\sqrt{5}}{2}$.
2. By using the barcode matrix, a one-to-one correspondence from the matrix and the DNA sequence has been given. In particular, the barcode picture (see *e.g.*, Figure 35.3) shows that the nucleotides are not homogeneously distributed and the probability distribution has been computed.

3. By using the cardinal complex representation (35.14), we can see that in the complete sequence (coding and noncoding regions), a long-range correlation exists (Figure 35.6) with $\alpha \cong 0.61$.
4. It has been shown, by a cluster analysis of the wavelet coefficients, that not only the wavelet coefficients of the cardinal complex representation but also of the random walk on which the cardinal complex series belong are displayed on some symmetrical grids (Figures 35.11, 35.13, 35.14).
5. Clusters of wavelet coefficients are sensible to anomalies in the sequence (Figures 35.13, 35.14) in the sense that any pathology can be detected easily by the existence of new spots.
6. DNA walks have the same long-range correlation of the cardinal representation (Figure 35.16)
7. Both wavelet coefficients of cardinal complex representation and of corresponding DNA walk range in discrete sets. The clusters of wavelets are distributed on some grids that show some axial symmetries for both (sequence and walk) (Figures 35.11, 35.17, 35.18).
8. It has been shown that a random sequence is characterized by clusters of wavelet coefficients distributed on symmetrical grids (Figure 35.12). However, the corresponding walk does not have a symmetrical grid distribution (Figure 35.19).

## REFERENCES

1. M. Altaiski, O. Mornev, and R. Polozov. Wavelet analysis of DNA sequence. *Genet Anal*, 12:165–168, 1996.
2. D. Anastassiou. Frequency-domain analysis of biomolecular sequence. *Bioinformatics*, 16(12):1073–1081, 2000.
3. A. Arneado, E. Bacry, P.V. Graves, and J.F. Muzy. Characterizing long-range correlations in DNA sequences from wavelet analysis. *Phys Rev Lett*, 74:3293–3296, 1995.
4. A. Arneado, Y.D'Aubenton-Carafa, E. Bacry, P.V. Graves, J.F. Muzy, and C. Thermes. Wavelet based fractal analysis of DNA sequences? *Phys D*, 96:291–320, 1996.
5. A. Arneado, Y. D'Aubenton-Carafa, B. Audit, E. Bacry, J.F. Muzy, and C. Thermes. What can we learn with wavelets about DNA sequences? *Phys A*, 249:439–448, 1998.
6. B. Audit, C. Vaillant, A. Arneodo,Y. d'Aubenton-Carafa, and C. Thermes. Long range correlations between DNA bending sites: Relation to the structure and dynamics of nucleosomes. *JMB J Mol Biol*, 316:903–918, 2002.
7. J.A. Berger, S.K. Mitra, M. Carli, and A. Neri. Visualization and analysis of DNA sequences using DNA walks. *J Franklin Inst*, 341:37–53, 2004.
8. P. Bernaola-Galván, R. Román-Roldán, J.L. Oliver. Compositional segmentation and long-range fractal correlations in DNA sequences. *Phys Rev E*, 55(5):5181–5189, 1996.
9. C.L. Berthelsen, J.A. Glazier, and M.H. Skolnick. Global fractal dimension of human DNA sequences treated as pseudorandom walks. *Phys Rev A*, 45(12):8902–8913, 1992.

10. B. Borstnik, D. Pumpernik, and D. Lukman. Analysis of apparent $1/f^{\alpha}$ spectrum in DNA sequences. *Europhys Lett*, 23:389–394, 1993.
11. S.V. Buldyrev, A.L. Goldberger, A.L. Havlin, C.-K. Peng, M. Simons, F. Sciortino, and H.E. Stanley. Long-range fractal correlations in DNA. *Phys Rev E*, 51:5084–5091, 1995.
12. C. Cattani. Haar wavelet based technique for sharp jumps classification. *Math Comput Model*, 39:255–279, 2004.
13. C. Cattani. Haar wavelets based technique in evolution problems. *Proc Estonian Acad Sci Phys Math*, 53(1):45–63, 2004.
14. C. Cattani and J.J. Rushchitsky. Wavelet and wave analysis as applied to materials with micro or nanostructure, series on advances in mathematics for applied sciences. *World Sci Singapore*, 74, 2007.
15. C. Cattani. Complex representation of DNA sequences. *Proceedings of the Bioinformatics Research and Development Second International Conference, BIRD 2008, Vienna, Austria,* volume 13 in *Communications in Computer and Information Science*, Springer-Verlag, Berlin 2008, pp. 528–537.
16. E.A. Cheever, D.B. Searls, W. Karanaratne, and G.C. Overton. Using signal processing techniques for DNA sequence comparison. *Proceedings of the 15th Annual Bioengineering Conference, Northeast*, 1989, pp. 173–174.
17. P. D. Cristea. Large scale features in DNA genomic signals. *Signal Process*, 83:871–888, 2003.
18. E. Coward. Equivalence of two Fourier methods for biological sequences. *J Math Biol*, 36:64–70, 1996.
19. I. Daubechies. *Ten Lectures on Wavelets*. SIAM, Philadelphia, PA, 1992.
20. G. Dodin, P. Vandergheynst, P. Levoir, C. Cordier, and L. Marcourt. Fourier and wavelet transform analysis, a tool for visualizing regular patterns in DNA sequences, *J Theor Biol*, 206:323–326, 2000.
21. J.P. Fitch and B. Sokhansanj. Genomic engineering: Moving beyond DNA sequence to function. *Proc IEEE,* 88(12):1949–1971, 2000.
22. M. A. Gates. Simpler DNA sequence representations. *Nature*, 316(219):137–142, 1985.
23. M.A. Gates. A simple way to look at DNA. *J Theor Biol*, 119:319–328, 1986.
24. H. Gee. A journey into the genome: what's there. *Nature*, 12, 2001. http://www.nature.com/nsu/010215/010215-3.html.
25. The Genome Data Base http://gdbwww.gdb.org/; Genome Browser, http://genome.ucsc.edu; European Informatics Institute, http://www.ebl.ac.uk; Ensembl, http://www.ensembl.org.
26. K. Hu, P. Ch. Ivanov, Z. Chen, P. Carpena, and H.E. Stanley. Effect of trends on detrended fluctuation analysis. *Phys Rev E*, 64:011114, 2001.
27. X.-Y. Jiang, D. Lavenier, and S.S.-T. Yau, Coding region prediction based on a universal DNA sequence representation method. *J Comput Biol*, 15(10):1237–1256, 2008.
28. E. Hamori and J. Ruskin. H curves, a novel method of representation of nucleotide series especially suited for long DNA sequences. *J Biol Chem*, 258(2):1318–1327, 1983.
29. H. Herzel, E.N. Trifonov, O.Weiss, and I. Grosse. Interpreting correlations in biosequences. *Phys A*, 249:449–459, 1998.
30. S. Karlin and V. Brendel. Patchiness and correlations in DNA sequence. *Science*, 259:677–680, 1993.

31. W. Li. The complexity of DNA: The measure of compositional heterogenity in DNA sequence and measures of complexity. *Complexity*, 3:33–37, 1997.
32. W. Li. The study of correlation structures of DNA sequences: A critical review. *Computer Chem*, 21(4):257–271, 1997.
33. W. Li and K. Kaneko. Long-range correlations and partial $1/f^{\alpha}$ spectrum in a noncoding DNA sequence. *Europhys Lett*, 17:655–660, 1992.
34. K.B. Murray, D. Gorse, and J.M. Thornton. Wavelet transform for the characterization and detection of repeating motifs. *J Mol Biol*, 316:341–363, 2002.
35. C.-K. Peng, S.V. Buldryev, A.L. Goldberg, S. Havlin, F. Sciortino, M. Simons, and H.E. Stanley. Long-range correlatins in nucleotide sequences. *Nature*, 356:168–170, 1992.
36. C.-K. Peng, S.V. Buldryev, S. Havlin, M. Simons, H.E. Stanley, and A.L. Goldberg. Mosaic organization of DNA nucleotides. *Phys Rev E*, 49:1685–1689, 1994.
37. D.B. Percival and A.T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge University Press, Cambridge, UK, 2000.
38. A.A. Tsonis, P. Kumar, J.B. Elsner, and P.A. Tsonis. Wavelet analysis of DNA sequences. *Phy Rev E*, 53:1828-1834, 1996.
39. P.P. Vaidyanathan and B.-J. Yoon. The role of signal-processing concepts in genomics and proteomics. *J Franklin Inst*, 341:111–135, 2004.
40. R.F. Voss. Evolution of long-range fractal correlations and $1/f$ noise in DNA base sequences. *Phy Rev Lett*, 68(25):3805–3808, 1992.
41. R.F. Voss. Long-range fractal correlations in DNA introns and exons. *Fractals,*, 2:1–6, 1992.
42. O. Weiss and H. Herzel. Correlations in protein sequences and property codes. *J Theor Biol*, 190:341–353, 1998.
43. G. Wornell. *Signal Processing with Fractals: A Wavelet-Based Approach*, Prentice Hall Upper Saddle River, NJ, 1996.
44. S.S.-T, Yau, J. Wang, A. Niknejad, C. Lu, N. Jin, and Y.-K. Ho. DNA sequence representation without degeneracy. *Nucleic Acids Res*, 31:3078–3080, 2003.
45. Z.G. Yu, W.W. Anh, and B. Wang. Correlation property of length sequences based on global structure of the complex genome. *Phys Rev E*, 63:011–903, 2001.
46. M. Zhang. Exploratory analysis of long genomic DNA sequences using the wavelet transform: Examples using polyomavirus genomes. *Genome Sequencing and Analysis Conference VI*, 1995, pp. 72–85.

# 36

# HAPLOTYPE INFERENCE MODELS AND ALGORITHMS

Ling-Yun Wu

## 36.1 INTRODUCTION

With complete genome sequences for humans and many organisms available in the postgenomics era, one of the most important tasks in biological and medical research is to identify the genes related to diseases. The genetic study to find the association between genes and common complex diseases needs to assign a phenotype to a genetic region that is identified by one or several markers such as a single nucleotide polymorphism (SNP), microsatellite. Haplotype is a set of neighboring SNPs (or other genetic markers) that are transmitted together on the same chromosome. Because they capture information about regions descended from ancestral chromosomes, haplotypes generally have higher power than individual markers in association studies of the common complex diseases [3, 80]. Haplotypes also play a very important role in other areas of genetics such as population history studies.

Although the haplotypes can be determined by the use of existing experimental techniques [68], the current laboratory approaches for obtaining haplotypes directly from DNA samples are considerably expensive and time consuming; therefore they are not practical [93, 98, 92]. Hence, computational methods that offer a way of inferring haplotypes from existing data (*e.g.*, DNA sequencing data and genotype data) become attractive alternatives. There are two classes of *in silico* haplotyping problems: the haplotype assembly problem and the haplotype inference problem. The haplotype assembly problem (also called the individual haplotyping problem) is to assemble the aligned SNP fragments from shotgun sequencing methodology,

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

whereas the haplotype inference problem (also called the population haplotyping problem) is to infer a set of haplotypes of a population from their genotype dataset. In this chapter, we will focus on the models and algorithms for the haplotype inference problem.

Current practical laboratory techniques easily can produce unphased genotype data (*i.e.*, an unordered pair of alleles for each marker, that is, we do not know which alleles come from the same chromosome). The reconstruction of haplotypes from genotype data is a fundamental and decisive step in genetic studies. According to the available information, currently, there are mainly two ways for solving the haplotype inference problem. The first approach is haplotyping genetically related individuals (*e.g.*, the people from a family). By exploiting the additional pedigree data, we can get a better estimate of haplotypes because the haplotype pair of a child is constrained by its inheritance from his parents. The disadvantage of this approach is that it involves significant additional genotyping costs and potential recruiting problems, and there are still a portion of the alleles that are ambiguous. The second approach is haplotyping a population without pedigree information. This fast and cheap population-based alternative applies computational or statistical methods to find the most likely haplotype configurations from the observed genotype data.

There are already some review papers (*e.g.*, Bonizzoni *et al.* [7], Gusfield [38], Halldörsson *et al.* [40], Stephens and Donnelly [81], Adkins [2], and Gao *et al.* [25]). But most of them focus on only some aspects of the haplotype inference problem. For example, Bonizzoni *et al.* [7], Gusfield [38], and Halldörsson *et al.* [40] mainly reviewed the combinatorial approaches. Stephens and Donnelly [81] and Adkins [2] focused on the population-based statistical methods, whereas Gao *et al.* [27] surveyed the statistical methods for pedigree data. In this chapter, we try to give the readers a brief but complete overview on the haplotype inference problems from modeling and algorithmic viewpoints, focusing on the recent research progress.

The rest of the chapter is organized as follows. In the next section, the problem statement of the haplotype inference problem is given. Then the models and algorithms are reviewed in three groups: the combinatorial methods in Section 36.3, the statistical methods in Section 36.4, and the methods for pedigree haplotype inference problem in Section 36.5. Finally, the evaluation and comparison of haplotype inference methods are introduced in Section 36.6, and some discussion is presented in the last section.

## 36.2 PROBLEM STATEMENT AND NOTATIONS

In this section, we first introduce the haplotype inference problem and related notations. For the diploidy organisms such as human, each individual has two nearly identical copies of each chromosome and hence of each region of interest. A description of the alleles of markers on a single copy is called a *haplotype*, whereas a description of the conflated alleles (*i.e.*, the unordered pair of alleles for each marker) on the two homogenous copies of chromosomes is called a *genotype*. For SNPs, the possible alleles in each site are 2, and the alleles often are labeled “0” (wild type) and “1” (mutant type). Then, mathematically, a haplotype $\boldsymbol{h}$ can be denoted as a vector $(h_1, \cdots, h_n)$ over $\{0, 1\}$. A genotype $\boldsymbol{g}$ denotes a vector $(g_1, \cdots, g_n)$ over $\{0, 1, 2\}$. Let $\boldsymbol{h}_1, \boldsymbol{h}_2$ be a pair of haplotypes from the corresponding pair of chromosomes. Then the relationship between $\boldsymbol{h}_1, \boldsymbol{h}_2$ and $\boldsymbol{g}$ is:

$$g_i = \begin{cases} 0, & \text{if } h_{1i} = h_{2i} \text{ and } h_{1i}, h_{2i} \text{ are wild} \\ 1, & \text{if } h_{1i} = h_{2i} \text{ and } h_{1i}, h_{2i} \text{ are mutant} \\ 2, & \text{if } h_{1i} \neq h_{2i} \text{ (\textit{i.e.}, the } i\text{th SNP site is heterozygous)} \end{cases} \tag{36.1}$$

A pair of haplotypes $\boldsymbol{h}_1, \boldsymbol{h}_2$ is called a *haplotype configuration* or a *resolution* of a genotype $\boldsymbol{g}$ if they satisfy Equation (36.1). We denote it by $\boldsymbol{h}_1 \oplus \boldsymbol{h}_2 = g$, $\boldsymbol{h}_1 = \boldsymbol{g} \ominus \boldsymbol{h}_2$, $\boldsymbol{h}_2 = \boldsymbol{g} \ominus \boldsymbol{h}_1$, where $\boldsymbol{h}_1, \boldsymbol{h}_2$ are said to *resolve* the genotype $\boldsymbol{g}$. A genotype may have many haplotype configurations ($2^{k-1}$ configurations if there are $k$ heterozygous sites). A genotype is called *ambiguous* if it has at least two heterozygous positions; otherwise, it is called *resolved.* A haplotype $\boldsymbol{h}$ is called *compatible* with a genotype $\boldsymbol{g}$ if for all $g_i \neq 2$, $h_i = g_i$. The genotype data in a population of size $m$ can be formulated as an $m \times n$ matrix $G = \{g_{ij}\}$ on $\{0, 1, 2\}$ with each row $\boldsymbol{g}_i$ corresponding to an genotype and each column $j$ corresponding to an SNP site on the chromosome. A *realization* of a genotype matrix is a haplotype matrix $H$ on $\{0, 1\}$ with each row corresponding to a haplotype, and for each genotype $\boldsymbol{g}_i$, there are two rows (a pair of haplotypes) $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ of $H$ such that $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ form a resolution of $\boldsymbol{g}_i$.

For a pair of chromosomes of a child, if no *recombination* occurs, then one copy is inherited identically from the paternal genome and the other is inherited from the maternal genome; otherwise, portions of the paternal or maternal chromosomes are exchanged during inheritance, as shown in Figure 36.1. Biological experiments [13]

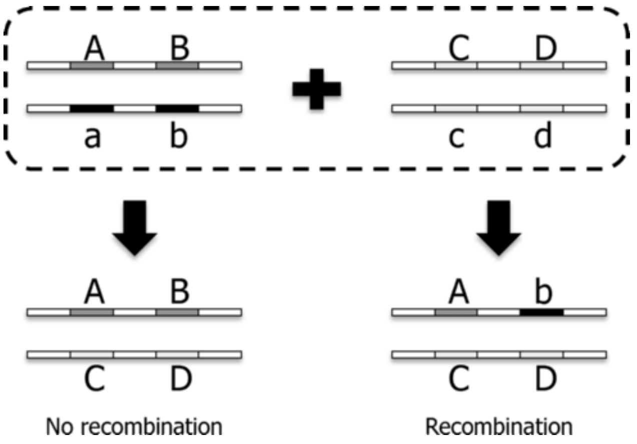


**Figure 36.1** Recombination event illustration. A recombination event occurs when the chromosomes of the right child are inherited from parents (SNP A and b are from different chromosomes of one parent), and no recombination occurs when the chromosomes of the left child are inherited from parents.

show that human chromosomes have a block structure called *linkage disequilibrium* (LD) block in which, within each block, no or few recombinations could occur and haplotypes have very low diversity. These facts make the haplotype mapping and disease association study possible. Haplotype inference is to resolve the heterozygous sites in a set of genotype data (*i.e.*, to determine which copy of a pair of chromosomes each allele belongs to). Mathematically, the haplotype inference problem is given as follows:

> Given an $m \times n$ genotype matrix $G$, find a haplotype matrix $H$ such that for each genotype at least one pair of haplotypes in $H$ exists that is a resolution of this genotype.

Obviously, without any biological insight or genetic model, we cannot infer haplotypes from genotype data because there may be an exponential number of possible haplotype configurations. If we arbitrarily select a pair of haplotypes among them for a genotype, then the haplotype inference problem is trivial and we do not know which one is "true." Blocks of limited haplotype diversity make haplotype inference somewhat easier than the case that many recombination events occur. Therefore, an inferring method mainly considers the analysis of a specific block in the population. Actually, the haplotype inference problem has several versions based on different genetic models and assumptions. The most powerful genetic model is the population-genetic concept of a *coalescent* [44, 85].

Since the first haplotype inference model, the Clark's rule, was proposed in 1990, a lot of approaches to population-based haplotyping problem were investigated. Roughly, these methods can be classified into two groups: the combinatorial methods and the statistical methods. The former is a deterministic approach that includes the parsimony methods and the phylogeny methods, and the latter method exploits stochastic tools. We are going to introduce the methods of each group one by one in the next sections.

## 36.3 COMBINATORIAL METHODS

### 36.3.1 Clark's Inference Rule

Clark [11] first pointed out some basic computational issues related to haplotype inference under a general *inference rule*. Suppose that $G$ is a set of genotypes with $n$ SNP sites. First, all the resolved genotypes in $G$ are identified and form a haplotype set $H$. For a genotype $\boldsymbol{g} \in G$, select a haplotype $\boldsymbol{h}$ compatible with $\boldsymbol{g}$ in $H$. If there is no $\boldsymbol{g} \ominus \boldsymbol{h}$ in $H$, then add $\boldsymbol{g} \ominus \boldsymbol{h}$ into $H$. Remove $\boldsymbol{g}$ from $G$. The process continues until either all genotypes are resolved or no haplotype in $H$ is compatible with the left genotypes. A greedy approach is applied when the known haplotypes are tested against the unresolved genotypes. A simple example is illustrated as follows:

### ■ EXAMPLE 36.1

Given a set of genotype *G:*

$$G = \begin{pmatrix} 2 & 1 & 2 & 1 & 1 & 2 \\ 0 & 1 & 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 2 & 0 & 2 \\ 2 & 2 & 0 & 2 & 2 & 0 \end{pmatrix}$$

The second genotype $g_2 = (0\ 1\ 2\ 1\ 1\ 0)$ is resolved because it has only one heterozygous site. Two haplotypes are identified as $h_1 = (0\ 1\ 0\ 1\ 1\ 0)$ and $h_2 = (0\ 1\ 1\ 1\ 1\ 0)$. Note that $h_1$ is compatible with $g_1 = (2\ 1\ 2\ 1\ 1\ 2)$; therefore, another haplotype is identified as $h_3 = g_1 \ominus h_1 = (1\ 1\ 1\ 1\ 1\ 1)$. $h_1$ is also compatible with $g_4 = (2\ 2\ 0\ 2\ 2\ 0)$; therefore, $h_4 = (1\ 0\ 0\ 0\ 0\ 0)$. Finally, $g_3 = (1\ 2\ 0\ 2\ 0\ 2)$ is compatible with $h_4$, so then $h_5 = g_3 \ominus h_4 = (1\ 1\ 0\ 1\ 0\ 1)$. Hence, five haplotypes are obtained as follows:

$$H = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

And the haplotype configurations for each genotype:

$$\begin{aligned} g_1 &= h_1 \oplus h_3 \\ g_2 &= h_1 \oplus h_2 \\ g_3 &= h_4 \oplus h_5 \\ g_4 &= h_1 \oplus h_4 \end{aligned}$$

Clark's experiments on real data and simulated data suggest that a valid solution is usually the one that resolves a maximum number of genotypes. Based on this observation, a lot of parsimony methods were proposed in the past two decades. Owing to its intuitive simplicity and biological implication, the parsimony methods were investigated extensively.

The maximum resolution (MR) model is based on Clark's inference rule and a parsimony principle: the real solution of haplotype inference can resolve a maximum number of genotypes. Then the haplotype inference problem becomes: Given a set of genotypes $G$ (including some resolved ones), find a maximum number of genotypes that can be resolved by successive application of Clark's inference rule.

The MR model can be formulated exactly as an integer linear programming and was proved to be nondeterministic polynomial (NP)-hard (Gusfield [35]). Gusfield

[34, 35] employed a graph-theoretical method to express and analyze the inference problem. Linear programming relaxation is adopted to solve the practical problem.

The single genotype resolution (SGR) model also is based on Clark's inference rule and is related to the MR problem. It was formulated first by Bonizzoni *et al.* [7]. Given a nonempty set of haplotypes $H$ and a distinguished genotype $\boldsymbol{g}$ in a set of genotypes $G$, this model finds a sequence of applications of the Clark's inference rule that resolves a subset $G_0 \subset G$ including $\boldsymbol{g}$ or conclude that such a sequence does not exist.

The computational complexity of the SGR problem was listed as an open problem by Bonizzoni *et al.* [7]. Lin *et al.* [59] solved it by reduction from 3 satisfiability (SAT) problem and thus proved that the SGR problem is NP-complete. So far, there is no algorithm designed for this problem.

### 36.3.2 Pure Parsimony Model

The MR model and the SGR model both are defined by using Clark's rule. Gusfield [37] further studied a parsimony version of the Clark's rule and proposed the pure parsimony criterion for haplotype inference, which does not rely directly on the Clark's rule. Its reasonability and biological meanings were illustrated in Gusfield [37], Wang and Xu [89]. This criterion is based on the fact that in natural populations, the number of the observed distinct haplotypes is vastly smaller than the number of combinatorially possible haplotypes. The haplotype inference by pure parsimony (HIPP) is defined as follows: Given a set of genotypes $G$, find a cardinality-smallest set of haplotypes $H$ such that for each $\boldsymbol{g} \in G$, there is a haplotype configuration consisting of two sequences in $H$ to resolve $\boldsymbol{g}$.

The HIPP problem is proved approximable (APX)-hard by Lancia *et al.* [50]. A branch-and-bound method was suggested by Wang and Xu [89]. An integer programming model of exponential size was presented in Gusfield [37], and later an integer programming model of linear size was given by Brown and Harrower [8]. The first approximation algorithm with performance guarantee $2^{k-1}$ was designed by Lancia *et al.* [50] for the case in which each genotype has at most $k$ heterozygous positions. Huang *et al.* [43] proposed an $O(\log n)$-approximation algorithm, where $n$ is the number of genotypes. A heuristic algorithm—parsimonious tree-grow method (PTG)—in $O(m^2n)$ time ($m$ is the number of SNP sites, and $n$ is the number of genotypes) was developed by Li *et al.* [58], which can not only solve haplotyping problem in an accurate and efficient manner but also numerically handle large-scale problems, and a software is also provided for PTG in http://zhangroup.aporc.org/bioinfo/ptg/. Recently, Lancia and Rizzi [51] presented a polynomial time algorithm for the HIPP problem when each genotype has at most two heterozygous positions.

The parsimony methods are developed from the Clark's rule [11]. The advantage of the parsimony models is intuitive simplicity; therefore, it is easy to understand. The parsimony criterion also has many biological implications and is used widely in biological research, especially in bioinformatics. But most models derived by parsimony criterion are difficult to solve. Some of them are NP-hard, whereas some are even APX-hard. Therefore, the parsimony methods are not practical for large-scale applications. For real biological data, the numbers of genotypes and SNPs could be as large as thousands to millions because of the rapid advance of genotyping techniques such as SNP array. Moreover, as shown in some benchmark studies [64], the performance of parsimony methods is not good as the statistical methods, which partially may be a result of its simple parsimony assumption.

### 36.3.3 Phylogeny Methods

Phylogeny methods for haplotype inference is based on the coalescent theory in genetics. In the phylogeny model, the evolutionary history of the haplotypes in the population can be described by a rooted tree, and each haplotype is a leaf of the tree. Another assumption of the phylogeny model is *infinite site*. That is to say, at most, one mutation occurs in a given site in the tree, and recurrent mutations are forbidden. Hence, the infinite site model is suitable for describing the evolutionary history without recombination.

Let $H$ be a set of haplotypes (an $m \times n$ matrix on $\{0, 1\}$). A *haplotype perfect phylogeny* for $H$ is a rooted tree with the following properties:

1. Each leaf in the tree denotes a distinct haplotype in $H$.
2. Each edge represents an SNP site with a mutation from 0 to 1, and each site is labeled by at most one edge.
3. For each haplotype labeled by a leaf of the tree, the unique path from the root to itself specifies all SNP sites with a value of 1 in this haplotype.

For example, given a set of haplotypes $H$ as follows, Figure 36.2 is a haplotype perfect phylogeny for $H$.

$$H = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Given a genotype matrix $G$, find a haplotype set $H$ such that for each $\boldsymbol{g} \in G$, there is a pair of haplotypes in $H$ acting as a resolution of $\boldsymbol{g}$ and $H$ has a haplotype perfect phylogeny; otherwise, conclude that such a matrix does not exist. This is called a perfect phylogeny haplotype (PPH) problem and was introduced by Gusfield [36] in RECOMB 2002.

The PPH problem is polynomially solvable. Gusfield [36] solved the PPH problem by transforming it into a graph realization problem. Gusfield showed that the solution can be determined if it is unique; otherwise, all solutions can be represented as a linear-space data structure, both in linear time. Though the algorithm is of

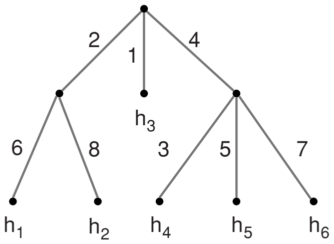


**Figure 36.2** An example of haplotype perfect phylogeny. The label of each edge is the index of SNP site, which is mutated from 0 to 1, and the labels $h_1$ to $h_6$ denote haplotypes in $H$.

polynomial time of $O(mn\alpha(mn))$ ($\alpha$ is the inverse Ackerman function), its realization procedure is very complicated. Two direct polynomial $O(mn\alpha(mn))$ algorithms based on "conflict-pairs" were proposed independently by Bafna *et al.* [87] and Eskin *et al.* [22]. Bafna *et al.* [4] claimed that these algorithms cannot be adapted to an algorithm of $O(mn)$, and any linear time solution to the PPH problem likely requires a different approach. Later, Gramm *et al.* [29] gave a linear time algorithm for PPH when the phylogeny is a path. Their algorithm relies on a reduction of the problem to that of deciding whether a partially ordered set has width 2. Ding *et al.* [18] solved this problem by giving a practical and deterministic $O(mn)$ algorithm based on a simple data structure and simple operatioins. The other two $O(mn)$ algorithms were proposed independently by Liu *et al.* [63] and Vijayasatya and Mukherjee [88].

If there are multiple solutions for PPH, then is it possible to find one that is most parsimonious in terms of the number of distinct haplotypes? Bafna *et al.* [4] proposed a variant of PPH problem called the minimum perfect phylogeny haplotype (MPPH) problem, which minimizes the number of distinct haplotypes in the PPH solutions. Bafna *et al.* [4] showed that the problem is NP-hard by a reduction from vertex cover. Recently, Sridhar *et al.* [79] developed the first practical method for reconstructing the minimum perfect phylogenies directly from genotype data. They showed that MPPH is computationally feasible for moderate-sized problem instances.

When there is inconsistency (read-errors, missing bases, or an imperfect fit to the perfect phylogeny model) in the data, PPH becomes a computationally hard problem. Gramm *et al.* [29] proved that PPH with missing data is NP-complete even when the phylogeny is a path and only one allele of every polymorphic site is present in the population in its homozygous state. Kimmel and Shamir [48] proved that the perfect phylogeny haplotype problem is NP-complete when some data entries are missing. Fortunately, Gramm *et al.* [30] found that haplotyping via perfect phylogeny with missing data becomes computationally tractable when imposing additional biologically motivated constraints. Similarly, Halperin and Karp [42] assumed a rich data hypothesis under which the problem becomes tractable.

Damaschke [14] proposed a different approach, which also is based on perfect phylogeny structure. Although most PPH algorithms are purely combinatorial and generate a description of all possible haplotyping results for any set of genotypes, Damaschke's approach concentrated on the haplotypes that can be inferred safely.

Damaschke [15] gave an algorithm that identifies haplotypes incrementally along the sequence, which also can recover missing data. Barzuza *et al.* [6] considered a variant of the PPH problem with less information (*i.e.*, only the heterozygosity or homozygosity information is known for genotypes).

The perfect phylogeny assumption may be too restrictive for real biological data. Data inconsistent with perfect phylogenies can come from multiple mutations, recombinations, gene conversions, and so on. Therefore, several new models have been taken into account more realistic molecular evolution. Halperin and Eskin [41] successfully used the imperfect phylogeny (IPPH) model to reconstruct long haplotypes. The imperfect phylogeny method was shown to be very fast and accurate. Song *et al.* [77] presented a polynomial-time algorithm for the case when a single site is allowed to mutate twice (or a single recombination is present) under a practical assumption on the input data. Sridhar *et al.* [78] showed that it is possible to infer imperfect phylogenies with any constant number $q$ of recurrent mutations in polynomial time in the number of sequences and number of sites typed. But the computational complexity of general IPPH is still an open problem.

## 36.4 STATISTICAL METHODS

### 36.4.1 Maximum Likelihood Methods

In most statistical models for haplotype inference, there is an underlying unknown distribution of the haplotype frequencies in the population. These models often assume the Hardy–Weinber equilibrium (HWE). That is, the probability of a genotype is related to the probabilities of its haplotype configurations. In detail, for a genotype $\boldsymbol{g} \in G$,

$$\Pr(\boldsymbol{g}) = \sum_{\boldsymbol{h} \oplus \bar{\boldsymbol{h}} = \boldsymbol{g}} \Pr(\boldsymbol{h}) \Pr(\bar{\boldsymbol{h}})$$

where $\Pr(\cdot)$ represents the probability. The maximum likelihood methods try to estimate the haplotype frequencies that maximize the likelihood function of the observed genotype set $G$. This problem also is known as *haplotype frequency estimation*.

An expectation-maximization (EM) algorithm was proposed by Excoffier and Slatkin [23]. This likelihood-based model does not make any assumption on the mutation and recombination. Note that because there is an exponential number of possible haplotypes, the EM algorithm cannot handle satisfactorily the problem with long haplotypes and large population size. In 2002, a partition-ligation-estimation-maximization (PLEM) algorithm developed by Qin *et al.* [70] uses a *divide and conquer* approach to address this issue.

Kimmel and Shamir [45, 47] extended the maximum likelihood method by explicitly considering the recombination. Their method simultaneously solves the haplotype inference problem and the haplotype block partitioning problem [13, 68, 96].

The haplotypes are derived through mutations from founder haplotypes in each haplotype block.

### 36.4.2 Bayesian Methods

Stephens *et al.* [83] proposed a modification of the maximum likelihood method by introducing an approximate population genetic model based on the coalescent theory to generate *a priori* distribution of the haplotype frequencies. Their algorithm samples new haplotypes for each genotype from the mosaics of all haplotypes currently assigned to the genotypes. The more the haplotypes are similar to the assigned haplotypes, the more likely they are to be chosen. Then they try to find the posterior distribution of the haplotype frequencies for a given genotype set $G$. This problem also is called *Bayesian haplotype inference*. Stephens *et al.* [83] solved the problem by a Markov chain Monte Carlo (MCMC) approach. A partition-ligation variant of the MCMC approach was proposed by Niu *et al.* [65] to deal with large-scale problems. Another difference is that the sampling strategy of Niu *et al.* [65] prioritize the haplotypes already assigned to many genotypes. Greenspan and Geiger [31] also presented a Bayesian network model explicitly considering the recombination.

In 2005, Stephens and Scheet [82] extended the Bayesian MCMC method by considering the decay of LD and the missing data in genotypes. In this model, a hidden Markov model (HMM) is constructed to sample haplotypes and estimate the haplotype frequencies, and the recombination processes are treated as "nuisance parameters." The algorithms in [80, 82] are implemented in the software PHASE, which often is considered the most accurate software for population-based haplotype inference so far [64]. But the running time of PHASE increases dramatically with the number of markers because of the adopted MCMC approach. Delaneau *et al.* [16] addressed this problem by using a binary tree representations to avoid searching in an exponential growth space. In their tests, the new algorithm can run up to 150 times faster than PHASE while maintaining similar accuracy.

### 36.4.3 Markov Chain Methods

Instead of estimating frequencies of full haplotypes as the previous models, the variable order Markov chain model proposed by Eronen *et al.* [20] estimates and uses frequencies of short haplotype fragments. The haplotype frequencies are calculated from the fragment frequencies by modeling the haplotype as a Markov chain of short fragments. These fragments are shorter regions potentially conserved for several generations and thus are more likely to be identified reliably in a population sample. This method is aimed at long marker maps, where LD between markers may be relatively weak. The Markov chain model can adapt better to the recombination because it does not assume haplotype blocks.

To find the haplotype configuration with maximum likelihood from a solution space with an exponential number of haplotypes, Eronen *et al.* use a heuristic partition-ligation method like that in Qin *et al.* [70]. However, the partition-ligation algorithm used in Eronen *et al.* [20] is only a greedy heuristic algorithm producing near-optimal haplotype reconstructions in reasonable time and without any guarantee of solution quality. Recently, Zhang *et al.* [94, 95] proved that this problem can be solved exactly by a dynamic programming algorithm with polynomial time complexity. The computational experiments in [94, 95] show that the solution of the partition-ligation method may be far away from true optimum in some cases such as large marker spacing. Therefore, a dynamic programming method greatly can improve the results in [20], whereas the time and space complexity remains in the same low magnitude of the partition-ligation method.

Eronen *et al.* [21] introduced an EM algorithm to improve iteratively the accuracy of fragment frequencies estimation. This algorithm reestimates the fragment frequencies from the combined set of the most probable haplotype configurations for all genotypes, which is sampled in the previous step. The experiments in Eronen *et al.* [21] show that the algorithms outperform most existing haplotype inference methods, especially on genetically long marker maps. Based on the work in [21], Wu *et al.* [91] show that, by using the dynamic programming method proposed in [94, 95], the fragment frequencies can be reestimated more accurately from the possible haplotype configurations for all genotypes, which further improves the accuracy of haplotype inference.

Browning and Browning [9] proposed another haplotype inference method based on the variable Markov chain model. They described a localized haplotype cluster model that can be interpreted as a special class of HMMs. A stochastic EM-like algorithm is used. The algorithm involves iterative fitting a localized haplotype model to estimated haplotype configurations and sampling haplotype estimates conditional on the fitted localized haplotype model and the genotype data.

Scheet and Stephens [75] proposed an HMM to model the mutation and recombination on haplotypes in the maximum likelihood model. This algorithm was implemented in the software fastPHASE. But unlike the models based on the variable Markov chain [21, 9, 91], the number of haplotype clusters (*i.e.*, the number of states in each marker) is predetermined so that it can not adapt locally to the haplotype structures. Kimmel and Shamir [46] also adopted the HMM in the maximum likelihood model for haplotype inference. Similarly, Landwehr *et al.* [53] proposed a method based on constrained HMMs.

Recently, Rastas *et al.* [72] presented a new approach that combines the variable Markov chain model and the Bayesian method in which the haplotype inference relies on the posterior distribution of haplotype assignments in the previous step like in PHASE [80, 82]. The Bayesian step is solved efficiently by using a so-called context tree weighting algorithm.

## 36.5 PEDIGREE METHODS

The previous part of this chapter introduced the haplotype inference approaches in an unrelated population. However, some methods also can be applied to pedigree data. In this section, we will focus on the models and methods specially designed for the haplotype inference with pedigree data. Generally, haplotype inference based on pedigree data has two fundamental assumptions:

1. The given genotype data has a pedigree structure called pedigree graph. That is to say, the individuals in the population are related genetically.
2. The inheritance satisfies the Mendelian law (*i.e.*, out of two alleles in every SNP site of the genotype of a child, one comes from his paternal genome and the other from his maternal genome), and there is no mutation to occur during the inheritance.

One then can get a better estimation of haplotypes because the haplotypes of a child is constrained by its inheritance from his parents. However, collection of such pedigree data (related individuals) costs much more than that of an unrelated population. The complexity of haplotype inference with pedigree data heavily depends on the structure of the pedigree graph.

### 36.5.1 Minimum Recombinant Haplotype Configurations

The first version of haplotype inference problem with pedigree data is the minimum recombinant haplotype configurations (MRHC) problem. The MRHC problem is defined as follows: Given a valid genotype pedigree graph $G$, find a realization $H$ of $G$ involving a minimum number of recombination events. This version of haplotype inference was studied in the literature [39, 84, 69, 55, 56, 57, 19]. The models are based on the fact that few recombinations occur when the haplotypes of a child are inherited from parents, and hence, the objective is often to minimize the total number of recombinations.

Haines [39] designed an algorithm to solve MRHC and used it to detect genotyping error. Tapadar *et al.* [84] presented a genetic algorithm based on certain principles of biological evolution. Qian and Bechkmann [69] presented a rule-based algorithm, which allows an exhaustive search of all possible haplotype configurations under the criterion that there are minimum recombinants between markers and can be applied to various pedigree structures. The algorithm performs well for small pedigrees but runs slowly on moderate or large-scale data. Li and Jiang [55] proved MRHC on a general pedigree graph to be NP-hard by reduction from three-dimensional matching and proposed an iterative heuristic algorithm. Doi *et al.* [19] proved MRHC on a pedigree tree is also NP-hard. They gave two dynamic programming algorithms for MRHC on pedigree tree. Although most algorithms do not consider the missing data, Li and Jiang [56, 57] developed an integer linear programming formulation of the MRHC problem with missing data and a branch-and-bound algorithm to solve it.

### 36.5.2 Zero Recombinant Haplotype Configurations

A widely studied variant version of MRHC is the zero recombinant haplotype configurations (ZRHC) problem: Given a valid genotype pedigree graph $G$, find a realization $H$ of $G$ involving no recombination events or decide that such a realization does not exist. The zero recombinant assumption in the ZRHC is a little different from that in the phylogeny methods for population-based haplotype inference. The assumption here requires that recombination events do not occur among the generations in a single pedigree, whereas the assumption in the phylogeny methods implies that there are no recombination events since the ancestral haplotypes of the whole population. Note that the timespan of a pedigree typically is much smaller than that of the evolution of a population from their common ancestral haplotypes. Therefore, the zero recombinant assumption in the ZRHC is more realistic.

Wijsman [90] presented a rule-based algorithm for ZRHC. O'Connell [66] developed a genotype-elimination [67, 54] algorithm, which use the recoded alleles to delete inconsistent genotypes and to find the all possible haplotype configurations. Li and Jiang [55] proved the ZRHC is polynomial time solvable and presented an algorithm based on Gaussian elimination for ZRHC. Based on a generalization of the genetic rules of Wijsman [90] and the genotype-elimination algorithm of O'Connell [66], Zhang *et al.* [97] designed the software HAPLORE in which the haplotype frequencies are estimated by a partition-ligation-expectation-maximization algorithm in [70]. The algorithm developed by Baruch *et al.* [5] addressed very large pedigrees and small chromosomal segments, assuming no recombination and allowing for missing data.

If we relax the strict constraints on the number of recombination events in the ZRHC, then we can get another variant of MRHC, $k$-minimum recombination haplotype configuration ($k$-MRHC), which is defined as follows: Given a valid genotype pedigree graph $G$, find a realization $H$ of $G$ such that the total number of recombinations is minimal and the number of recombinations on each parent-offspring pair is at most $k$. Chin *et al.* [10] proved $k$-MRHC on a pedigree graph to be NP-hard even for $k = 1$. They proposed a dynamic programming in $O(nm_0^{3k+1}2^{m_0})$ time on pedigrees with $n$ nodes and at most $m_0$ heterozygous loci in each node. The computational complexity of $k$-MRHC on a pedigree tree is still open. Several variants of MRHC also were formulated by Bonizzoni *et al.* in their review paper [7]. The complexity and algorithms of these variants mostly are open.

### 36.5.3 Statistical Methods

The statistical methods for population-based haplotype inference are often flexible enough to be adapted to use pedigree information such as nuclear family and trios. For example, the methods in [82, 41] were modified to exploit the trios pedigree information in [64]. These methods often assume HWE in population and no recombination in pedigrees and extend the population-based approaches by excluding the parental haplotypes that are not consistent with the children's genotype data. Unlike the methods designed for pedigrees, these methods only infer the haplotypes of parents instead of whole families. Similar works can be found in [60, 73, 17, 62].

There are also several statistical methods specially designed for the pedigree-based haplotype inference problem [76, 61, 86]. Most are maximum likelihood methods with HWE assumption. Kruglyak *et al.* [49] applied the Lander–Green algorithm [52] and the Viterbi algorithm [71] to reconstruct haplotype configurations. Gudbjartsson *et al.* [32, 33] and Abecasis *et al.* [1] also used similar approaches. Fishelson *et al.* [24] developed a maximum likelihood method based on the Bayesian network. Although these are exact algorithms, the approximation methods are needed for large and complex pedigrees with large size of markers. Sobel *et al.* [76] presented an MCMC method based on simulated annealing. Lin and Speed [61] proposed another MCMC method based on a Gibbs-jump algorithm. Thomas *et al.* [86] designed a block Gibbs sampler for haplotype linkage analysis. Gao *et al.* [28] and Gao and Hoeschele [27] presented the conditional enumeration method, a deterministic approximation. Because the likelihood-based methods often ignore LD between markers, they are suitable for linkage analysis in long chromosomal regions with low-density markers but less accuracy in haplotype inference. Moreover, although these methods are capable of haplotyping, most focus on linkage analysis instead of haplotyping. The implementation of some methods do not provide a haplotyping output. A detailed review on statistical haplotype inference methods for pedigree data can be found in Gao *et al.* [25].

## 36.6 EVALUATION

### 36.6.1 Evaluation Measurements

There are several widely used measurements to evaluate the computational results of haplotype inference methods. The switch error, incorrect genotype percentage (IGP) and incorrect haplotype percentage (IHP) are measurements to compare the inferred haplotype configurations with the true haplotype configurations. *Switch error* is the percentage of possible switches ("recombinations") used to recover the correct haplotype configuration of an individual [60]. Switch error is a natural error measure for the haplotype inference problem because many applications using inferred haplotypes will look at local haplotype segments, and they are correct unless one of the needed switches is within the segment. IGP is the percentage of sites with incorrect inferred phase when the putative haplotypes are aligned with the real haplotypes to minimize the phase differences. IHP is the percentage of ambiguous individuals of which the inferred haplotype configuration is not completely correct [83]. The switch error, IGP, and IHP typically are used for high-density markers (*e.g.*, in a short chromosomal region.)

However, for low-density markers, the likelihood of the haplotype configurations and the effects of haplotype configurations on the accuracy of association studies such as quantitative trait locus (QTL) mapping is considered more important [26]. The numbers of recombinant in the inferred haplotype configurations also often are used for the rule-based methods [19, 39, 55, 56, 57, 69, 84]. The aims of phylogeny methods include the construction of phylogeny besides the haplotype inference; therefore, the accuracy of inferred haplotype is not the most important criteria of assessment. The authors of phylogeny methods claim that phylogeny reconstruction directly from unphased data is computationally feasible for moderate-sized problem instances and can lead to substantially more accurate tree size inferences than the standard practice of treating phasing and phylogeny construction as two separate analysis stages [5, 55].

### 36.6.2 Comparisons

There is no unified framework for the haplotype inference problem. There are two parallel main methodologies in inference model formulation; one is deterministic and the other is statistic. Various inference models have been suggested by featuring different computational complexity ranging from polynomial (P) to NP. These models of haplotype inference have different biological assumptions. Therefore, theoretical comparison of methods with different assumptions is difficult if not impossible. Some researchers have done comparisons of haplotype inference methods based on similar models. For example, Stephens and Donnelly [81] compared three Bayesian methods, emphasizing the differences between the models and the computational strategies.

Numerical comparison of models and algorithms need large and objective datasets because every model has its suited data. Extensive and objective accuracy comparison of them is heavily technical but an important research work. There are already some attempts. Adkins [2] compared the accuracy of some methods using a large set of 308 empirically determined haplotypes based on 15 SNPs. Marchini *et al.* [64] conducted a comprehensive comparison of five leading algorithms for haplotype inference. The algorithms were applied to both father-mother-child trios data and unrelated data. Two kinds of datasets were used: the simulated data generated by a coalesent model and the real data produced from the publicly available HapMap [12] data.

Although most papers compared their works with other methods in the papers, these comparisons are sometimes unfair and not objective. First, these comparisons are incomplete. Only a few methods are chosen for comparison. Second, the selection of datasets has bias. The performance of haplotype inference methods depends on the underlying properties of the genotype data (*e.g.*, the density of markers). Therefore, large number of varying data sets are necessary for comparing methods from a complete viewpoint. Third, the parameters of other methods are not well tuned. Most authors just use the default parameter settings provided by the software without any tuning for the test data. Evaluating all existing inference models on one framework and on unified datasets is still a problem for future study.

### 36.6.3 Datasets

Currently, most works in literature used both simulated data and real data for evaluation purposes. The data can be simulated in different levels. For example, some researchers use real haplotype and simulate genotypes by randomly mating. This is consistent with the HWE assumption, which is used in most models. Furthermore, the haplotype itself can be simulated instead of using real data. Some haplotype data are simulated by using software with a coalesent model such as COSI [74], whereas some are simulated by simple probabilistic distribution (*e.g.*, uniform distribution). The haplotype inference methods based on coalesent theory such as PHASE [80, 82] may benefit from the data generated using coalesent model because the assumption of inference methods is consistent with the simulated data. This often is argued by the authors of the haplotype inference methods that do not use a coalesent model. In fact, it is difficult to design a completely fair simulation scheme for all haplotype inference methods.

Compared with the simulated data, the real data is less arguable. The public Daly set [13] is a real genotype set with missing data from a European-derived population. Daly set was used widely in many literature such as Eronen *et al.* [20, 21], Zhang *et al.* [95], and Stephens and Scheet [82]. The data consists of 103 SNPs ranging over 500 kb on chromosome 5q31 (Crohn's disease). Another important source of real data is the HapMap data, which consists of genotypes from four populations: 30 trios from the Yoruba population in Ibadan, Nigeria (YRI), 30 trios from the CEPH (Utah residents with ancestry from northern and western Europe) population (CEU), 45 unrelated individuals from China (CHB), and 45 unrelated individuals from Japan (JPT). Marchini *et al.* [64] constructed haplotype and genotype datasets from HapMap [12] data, which consists of about 1.22 million SNPs. The simulated and real data used in [64] can be downloaded from the Internet.[1] Browning and Browning [9] and Rastas *et al.* [72] also generated genotype datasets from the HapMap data. Because the performance of methods may vary on the datasets with different density of markers (*i.e.*, the distance between markers), both sparse and dense datasets are needed for evaluating haplotype inference methods. The sparse datasets can be filtered by removing some markers with a small minimum allele frequency and then selected randomly from the markers in dense datasets [64, 72].

## 36.7 DISCUSSION

In this chapter, we briefly introduced the haplotype inference problem and the recent progress of haplotype inference methods. From the type of used genotype data, the haplotype inference methods are classified into two categories: population-based and pedigree-based. From the methodology viewpoint, the haplotype inference methods can be divided into two groups: combinatorial methods and statistical methods. But this classification is not strict. When statistical models are adopted for haplotype inference, probability and stochastic process theory play main roles in establishing models and designing algorithms. But one also can find in the literature that an efficient algorithm may require both deterministic and stochastic techniques, depending on the problem setting. Because there are an exponential number of feasible solutions in the haplotyping problem, to design an efficient computational method with a high accuracy is still an important task. In this sense, statistical methods are also kind of combinatorial methods.

[1] See http://www.stats.ox.ac.uk/marchini/phaseoff.html.

LD decays is one of the most important properties of haplotype. To model the LD decays with distance, many approaches are developed in the literature: the coalescent model, haplotype block structure, size-varied sliding window, HMM, frequent haplotype fragments, and so on. The combinatorial methods often do not fully use the information of intermarker distances and are more appropriate for tightly linked markers in a small chromosomal region [25]. In practice, the optimal solution of many haplotype inference models does not necessarily correspond to "true" haplotypes with 100% accuracy. Incorporating more information into existing models may improve the accuracy of haplotype inference, which will be a future research direction.

## REFERENCES

1. G.R. Abecasis, S.S. Cherny, W.O. Cookson, and L.R. Cardon. Merlin—rapid analysis of dense genetic maps using sparse gene flow trees. *Nat Genet*, 30:97–101, 2002.
2. R.M. Adkins. Comparison of the accuracy of methods of computational haplotype inference using a large empirical dataset. *BMC Genet*, 5:22, 2004.
3. J. Akey, L. Jin, and M. Xiong. Haplotypes vs single marker linkage disequilibrium tests: What do we gain? *Eur J Hum Genet*, 9(4):291–300, 2001.
4. V. Bafna, D. Gusfield, S. Hannenhalli, and S. Yooseph. A note on efficient computation of haplotypes via perfect phylogeny. *J Comput Biol*, 11(5):858–866, 2004.
5. E. Baruch, J.I. Weller, M. Cohen, M. Ron, and E. Seroussi. Efficient inference of haplotypes from genotypes on a large animal pedigree. *Genetics*, 172:1757–1765, 2006.
6. T. Barzuza, J.S. Beckmann, R. Shamir, and I. Pe'er. Computational problems in perfect phylogeny haplotyping: Typing without calling the allele. *IEEE/ACM Trans Comput Biol Bioinform*, 5:101–109, 2008.
7. P. Bonizzoni, G.D. Vedova, R. Dondi, and J. Li. The haplotyping problem: An overview of computational models and solutions. *J Comput Sci Technol*, 18(6):675–688, 2003.
8. D.G. Brown and I.M. Harrower. A new integer programming formulation for the pure parsimony problem inhaplotype analysis. *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics (WABI)*, 2004, pp. 254–265.
9. S.R. Browning and B.L. Browning. Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *Am J Hum Genet*, 81:1084–1097, 2007.
10. F.Y. Chin, Q.F. Zhang, and H. Shen. *k*-recombination haplotype inference in pedigrees. *Proceedings of the International Conference on Computational Science (ICCS), LNCS 3515*, Springer-Verlag, Berlin, 2005, pp. 985–993.
11. A.G. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Mol Biol Evol*, 7(2):111–122, 1990.
12. The International HapMap Consortium. The international hapmap project. *Nature*, 426(6968):789–796, 2003.
13. M.J. Daly, J.D. Rioux, S.F. Schaffner, T.J. Hudson, and E.S. Lander. High-resolution haplotype structure in the human genome. *Nat Genet*, 29(2):229–232, 2001.

14. P. Damaschke. Fast perfect phylogeny haplotype inference. *Proceedings of the 14th Symposium on the Fundamentals of Computation. Theory FCT 2003, LNCS 2751*, 2003, pp. 183–194.
15. P. Damaschke. Incremental haplotype inference, phylogeny and almost bipartite graphs. *Proceedings of 2nd RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotypes*, 2004, pp. 1–11.
16. O. Delaneau, C. Coulonges, and J.-F. Zagury. Shape-it: New rapid and accurate algorithm for haplotype inference. *BMC Bioinformatics*, 9:540, 2008.
17. X. Ding, Q. Zhang, C. Flury, and H. Simianer. Haplotype reconstruction and estimation of haplotype frequencies from nuclear families with only one parent available. *Hum Hered*, 62:12–19, 2006.
18. Z. Ding, V. Filkov, and D. Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem. *J Comput Biol*, 13(2):522–553, 2006.
19. K. Doi, J. Li, and T. Jiang. Minimum recombinant haplotype configuration on tree pedigrees. *Proceedings of the 3th Annual International Workshop on Algorithms in Bioinformatics (WABI)*, Springer-Verlag, New York, 2003, pp. 339–353.
20. L. Eronen, F. Geerts, and H. Toivonen. A markov chain approach to reconstruction of long haplotypes. *Proceedings of the 9th Pacific Symposium on Biocomputing (PSB'04)*, World Scientific, Singapore, 2004, pp. 104–115.
21. L. Eronen, F. Geerts, and H. Toivonen. Haplorec: Efficient and accurate large-scale reconstruction of haplotypes. *BMC Bioinformatics*, 7(542), 2006.
22. E. Eskin, E. Halperin, and R.M. Karp. Efficient reconstruction of haplotype structure via perfect phylogeny. *J Bioinform Comput Biol*, 1(1):1–20, 2003.
23. L. Excoffier and M. Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Mol Biol Evol*, 12(5):921–927, 1995.
24. M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. *Hum Hered*, 59:41–60, 2005.
25. G. Gao, D.B. Allison, and I. Hoeschele. Haplotyping methods for pedigrees. *Hum Hered*, 67(4):248–266, 2009.
26. G. Gao and I. Hoeschele. Approximating identity-by-descent matrices using multiple haplotype configurations on pedigrees. *Genetics*, 171(1):365–376, 2005.
27. G. Gao and I. Hoeschele. A rapid conditional enumeration haplotyping method in pedigrees. *Genet Sel Evol*, 40:25–36, 2008.
28. G. Gao, I. Hoeschele, P. Sorensen, and F. Du. Conditional probability methods for haplotyping in pedigrees. *Genetics*, 167(4):2055–2065, 2004.
29. J. Gramm, T. Nierhoff, R. Sharan, and T. Tantau. On the complexity of haplotyping via perfect phylogeny. *Proceedings of Second RECOMB Satellite Workshop on Computational Methods for SNPs and Haplotypes*, Pittsburgh, PA, 2004, pp. 35–46.
30. J. Gramm, T. Nierhoff, and T. Tantau. Perfect path phylogeny haplotyping with missing data is fixed-parametertractable. In *Lecture Notes in Computer Science*, volume 3162. Springer-Verlag, New York, 2004, pp. 174–186.
31. G. Greenspan and D. Geiger. Model-based inference of haplotype block variation. *Proceedings of the 7th Annual International Conference on Computational Molecular Biology (RECOMB)*, 2003, pp. 131–137.
32. D.F. Gudbjartsson, K. Jonasson, M.L. Frigge, and A. Kong. Allegro, a new computer program for multipoint linkage analysis. *Nat Genet*, 25:12–13, 2000.

33. D.F. Gudbjartsson, T. Thorvaldsson, A. Kong, G. Gunnarsson, and A. Ingolfsdottir. Allegro version 2. *Nat Genet*, 37:1015–1016, 2005.

34. D. Gusfield. A practical algorithm for optimal inference of haplotypes from diploid populations. *Proceedings of the Eighth International Conference on Intelligent Systemsfor Molecular Biology (ISMB)*, AAAI Press, Menlo Park, CA, 2000, pp. 183–189.

35. D. Gusfield. Inference of haplotypes from samples of diploid populations: Complexity and algorithms. *J Comput Biol*, 8(3):305–323, 2001.

36. D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. *Proceedings of 6th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, ACM Press, Menlo Park, CA, 2002, pp. 166–175.

37. D. Gusfield. Haplotyping by pure parsimony. *Proceedings of the 14th Symposium on Combinatorial Pattern Matching (CPM)*, Springer-Verlag, New York, 2003, pp. 144–155.

38. D. Gusfield. An overview of combinatorial methods for haplotype inference. *Proceedings of the 1st RECOMB Satellite Workshop on Computational Methodsfor SNPs and Haplotype Inference*. Springer-Verlag, New York, 2004, pp. 9–25.

39. J.L. Haines. Chromlook: An interactive program for error detection and mapping in reference linkage data. *Genomics*, 14:517–519, 1992.

40. B.V. Halldórsson, V. Bafna, N. Edwards, R. Lippert, S. Yooseph, and S. Istrail. A survey of computational methods for determining haplotypes. In S. Istrail, M. Waterman, and A. Clark, editors, *SNPs and Haplotype Inference, LNBI 2983*. Springer-Verlag, Berlin Germany, 2004, pp. 26–47.

41. E. Halperin and E. Eskin. Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, 20(12):1842–1849, 2004.

42. E. Halperin and R.M. Karp. Perfect phylogeny and haplotype assignment. *Proceedings of 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 2004, pp. 10–19.

43. Y.-T. Huang, K.-M. Chao, and T. Chen. An approximation algorithm for haplotype inference by maximum parsimony. *Proceedings of the 2005 ACM Symposium on Applied Computing*, 2005, pp. 146–150.

44. R. Hudson. Gene genealogies and the coalescent process. *Oxf Sur Evol Biol*, 7:1–44, 1990.

45. G. Kimmel and R. Shamir. Maximum likelihood resolution of multi-block genotypes. *Proceedings of the 8th Annual International Conference on Computational Molecular Biology (RECOMB)*, 2004.

46. G. Kimmel and R. Shamir. A block-free hidden markov model for genotypes and its application to disease association. *J Comput Biol*, 12(10):1243–1260, 2005.

47. G. Kimmel and R. Shamir. Gerbil: Genotype resolution and block identification using likelihood. *Proc Natl Acad Sci U S A*, 102(1):158–162, 2005.

48. G. Kimmel and R. Shamir. The incomplete perfect phylogeny haplotype problem. *J Bioinform Comput Biol*, 3(2):359–384, 2005.

49. L. Kruglyak, M.J. Daly, M.P. Reeve-Daly, and E.S. Lander. Parametric and nonparametric linkage analysis: A unified multipoint approach. *Am J Hum Genet*, 58:1347–1363, 1996.

50. G. Lancia, C. Pinotti, and R. Rizzi. Haplotyping population by pure parsimony: Complexity of exact and approximationalgorithms. *INFORMS J Comput*, 16(4):348–359, 2004.

51. G. Lancia and R. Rizzi. A polynomial case of the parsimony haplotyping problem. *Oper Res Let*, 34(3):289–295, 2006.

52. E.S. Lander and P. Green. Construction of multilocus genetic linkage maps in humans. *Proc Nat Acad Sci*, 84:2363–2367, 1987.

53. N. Landwehr, T. Mielik0Ł1inen, L. Eronen, H. Toivonen, and H. Mannila. Constrained hidden markov models for population-based haplotyping. *BMC Bioinformatics*, 8(Suppl 2):S9, 2007.

54. K. Lange and T.M. Goradia. An algorithm for automatic genotype elimination. *Am J Hum Genet*, 40:250–256, 1987.

55. J. Li and T. Jiang. Efficient inference of haplotypes from genotypes on a pedigree. *J Bioinform Comput Biol*, 1(1):41–69, 2003.

56. J. Li and T. Jiang. An exact solution for finding minimum recombinant haplotype configurations on pedigrees with missing data by integer linear programming. *Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, ACM press, Menlo Park, CA, 2004, pp. 20–29.

57. J. Li and T. Jiang. Computing the minimum recombinant haplotype configuration from incomplete genotype data on a pedigree by integer linear programming. *J Comput Biol*, 12(6):719–739, 2005.

58. Z. Li, W. Zhou, X.-S. Zhang, and L. Chen. A parsimonious tree-grow method for haplotype inference. *Bioinformatics*, 21(17):3475–3481, 2005.

59. H. Lin, Z. Zhang, Q. Zhang, D. Bu, and M. Li. A note on the single genotype resolution problem. *J Comput Sci Technol*, 19(2):254–257, 2004.

60. S. Lin, A. Chakravarti, and D. Cutler. Haplotype and missing data inference in nuclear families. *Genome Res*, 14:1624–1632, 2004.

61. S. Lin and T.P. Speed. An algorithm for haplotype analysis. *J Comput Biol*, 4:535–546, 1997.

62. P.Y. Liu, Y. Lu, and H.W. Deng. Accurate haplotype inference for multiple linked single-nucleotide polymorphisms using sibship data. *Genetics*, 174:499–509, 2006.

63. Y. Liu and C.-Q. Zhang. A linear solution for haplotype perfect phylogeny problem. *International Conference on Bioinformatics and Its Applications (ICBA)*, 2004.

64. J. Marchini, D. Cutler, N. Patterson, M. Stephens, E. Eskin, E. Halperin, S. Lin, Z.S. Qin, H.M. Munro, G.R. Abecasis, *et al.* A comparison of phasing algorithms for trios and unrelated individuals. *Am J Hum Genet*, 78(3):437–450, 2006.

65. T. Niu, Z.S. Qin, X. Xu, and J.S. Liu. Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms. *Am J Hum Genet*, 70(1):157–169, 2002.

66. J.R. O'Connell. Zero-recombinant haplotyping: Applications to fine mapping using SNPs. *Genet Epidemiol*, 19(Suppl 1):S64–S70, 2000.

67. J.R. O'Connell and D.E. Weeks. An optimal algorithm for automatic genotype elimination. *Am J Hum Genet*, 65:1733–1740, 1999.

68. N. Patil, A.J. Berno, D.A. Hinds, W.A. Barrett, J.M. Doshi, C.R. Hacker, C.R. Kautzer, D.H. Lee, C. Marjoribanks, D.P. McDonough, *et al.* Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294(5547):1719–1723, 2001.

69. D. Qian and L. Beckmann. Minimum-recombinant haplotyping in pedigrees. *Am J Hum Genet*, 70(6):1434–1445, 2002.

70. Z.S. Qin, T. Niu, and J.S. Liu. Partition-ligation-expectation-maximization algorithm for haplotype inference with single-nucleotide polymorphisms. *Am J Hum Genet*, 71(5):1242–1247, 2002.

71. L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc IEEE*, 77(2):257–286, 1989.

72. P. Rastas, J. Kollin, and M. Koivisto. Fast bayesian haplotype inference via context tree weighting. *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics*, 2008.

73. K. Rohde and R. Fuerst. Haplotyping and estimation of haplotype frequencies for closely linked biallelic multilocus genetic phenotypes including nuclear family information. *Hum Mutation*, 17:289–295, 2001.

74. S.F. Schaffner, C. Foo, S. Gabriel, D. Reich, M.J. Daly, and D. Altshuler. Calibrating a coalescent simulation of human genome sequence variation. *Genome Res*, 15(11):1576–1583, 2005.

75. P. Scheet and M. Stephens. A fast and flexible statistical model for large-scale population genotype data: Applications to inferring missing genotypes and haplotypic phase. *Am J Hum Genet*, 78(4):629–644, 2006.

76. E. Sobel, K. Lange, J.R. O'Connell, and D.E. Weeks. Haplotyping algorithms. In T. Speed and M.S. Waterman, editors, *Genetic Mapping and DNA Sequencing*, volume 81 of *IMA Volumes in Mathematics and Its Applications*, Springer-Verlag, New York, 1996, pp. 89–110.

77. Y. Song, Y. Wu, and D. Gusfield. Algorithms for imperfect phylogeny haplotyping with a single homoplasy or recombination event. *Proceedings of Workshop on Algorithms in Bioinformatics*, 2005.

78. S. Sridhar, G.E. Blelloch, R. Ravi, and R. Schwartz. Optimal imperfect phylogeny reconstruction and haplotyping (ipph). *Computational System Bioinformatics Conference*, 2006, pp. 199–210.

79. S. Sridhar, F. Lam, G.E. Blelloch, R. Ravi, and R. Schwartz. Direct maximum parsimony phylogeny reconstruction from genotype data. *BMC Bioinformatics*, 8:472, 2007.

80. J.C. Stephens, J.A. Schneider, D.A. Tanguay, J. Choi, T. Acharya, S.E. Stanley, R. Jiang, C.J. Messer, A. Chew, J.H. Han, *et al*. Haplotype variation and linkage disequilibrium in 313 human genes. *Science*, 293(5529):489–493, 2001.

81. M. Stephens and P. Donnelly. A comparison of bayesian methods for haplotype reconstruction from population genotype data. *Am J Hum Genet*, 73(5):1162–1169, 2003.

82. M. Stephens and P. Scheet. Accounting for decay of linkage disequilibrium in haplotype inference and missing-data imputation. *Am J Hum Genet*, 76(3):449–462, 2005.

83. M. Stephens, N.J. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *Am J Hum Genet*, 68(4):978–989, 2001.

84. P. Tapadar, S. Ghosh, and P.P. Majumder. Haplotyping in pedigrees via a genetic algorithm. *Human Heredity*, 50(1):43–56, 2000.

85. S. Tavare. Calibrating the clock: Using stochastic processes to measure the rate of evolution. In E. Lander and M. Waterman, editors, *Calculating the Secrets of Life*. National Academy Press, Washington, DC, 1995.

86. A. Thomas, A. Gutin, V. Abkevich, and A. Bansal. Multilocus linkage analysis by blocked Gibbs sampling. *Stat Comput*, 10:259–269, 2000.

87. V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph. Haplotyping as perfect phylogeny: A direct approach. *J Comput Biol*, 10(3–4):323–340, 2003.

88. R. Vijayasatya and A. Mukherjee. An optimal algorithm for perfect phylogeny haplotyping. *J Comput Biol*, 13(4):897–928, 2006.

89. L. Wang and Y. Xu. Haplotype inference by maximum parsimony. *Bioinformatics*, 19(14):1773–1780, 2003.

90. E. Wijsman. A deductive method of haplotype analysis in pedigrees. *Am J Hum Genet*, 41:356–373, 1987.

91. L.-Y. Wu, J.-H. Zhang, and R. Chan. Improved approach for haplotype inference based on Markov chain. *Proceedings of the 2nd International Symposium on Optimization and Systems Biology (OSB 2008)*, volume 9, Beijing, China. World Publishing Corporation, Xian, China, 2008, pp. 204–215.

92. J. Xu. Extracting haplotypes from diploid organisms. *Curr Issues Mol Biol*, 8(2):113–122, 2006.

93. H. Yan, N. Papadopoulos, G. Marra, C. Perrera, J. Jiricny, C.R. Boland, H.T. Lynch, R.B. Chadwick, A. de la Chapelle, K. Berg, *et al.* Conversion of diploidy to haploidy. *Nature*, 403(6771):723–724, 2000.

94. J.-H. Zhang, L.-Y. Wu, J. Chen, and X.-S. Zhang. A new statistical method for haplotype inference from genotype data. *Proceedings of IASTED International Conference on Computational and Systems Biology (CASB 2006)*, ACTA Press, Calgary, Canada, 2006, pp. 7–12.

95. J.-H. Zhang, L.-Y. Wu, J. Chen, and X.-S. Zhang. A fast haplotype inference method for large population genotype data. *Comput Stat Data Anal*, 52(11):4891–4902, 2008.

96. K. Zhang, M. Deng, T. Chen, M.S. Waterman, and F. Sun. A dynamic programming algorithm for haplotype block partitioning. *Proc Natl Acad Sci U S A*, 99(11):7335–7339, 2002.

97. K. Zhang, F. Sun, and H. Zhao. HAPLORE: A program for haplotype reconstruction in general pedigrees without recombination. *Bioinformatics*, 21:90–103, 2005.

98. K. Zhang, J. Zhu, J. Shendure, G.J. Porreca, J.D. Aach, R.D. Mitra, and G.M. Church. Long-range polony haplotyping of individual human chromosome molecules. *Nat Genet*, 38(3):382–387, 2006.

# VII

# ANALYSIS OF BIOLOGICAL NETWORKS

# 37

# UNTANGLING BIOLOGICAL NETWORKS USING BIOINFORMATICS

Gaurav Kumar, Adrian P. Cootes, and Shoba Ranganathan

## 37.1 INTRODUCTION

### 37.1.1 Predicting Biological Processes: A Major Challenge to Understanding Biology

Cells are the building blocks of life in an organism. Each cell holds genes containing the information about constituent cellular processes. This genetic information is carried out by thousands of different proteins. Proteins are the nanomachines or tools of the cell that rarely work in isolation. They interact and constantly communicate with each other. The advent of genome sequencing projects has made available nearly a complete list of genetic information from the different kingdoms of life. However, this information does not give any direct insight into how the molecular components within the cell interact, resulting in the complexity of life forms. To understand cellular complexity, one has to map the molecular interactions occurring within the cell. Interaction maps provide guides on how different cellular entities like proteins, metabolites, and genes are linked together *via* direct physical or functional associations, to create the complex web of life.

The advent of high-throughput screening techniques has allowed the large-scale identification of components such as genes, RNAs, proteins, and metabolites. Although the data from such large-scale experimental studies are often incomplete and contain errors, they nonetheless provide valuable information about the function of

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

individual components and cellular processes. Thus, the large-scale datasets generated experimentally have provided input to different biological network models. The descriptions of biological networks are based on the mathematical foundations of network theory. We have therefore provided the core concepts of network features and principles, resulting from the statistical analysis of many man-made systems such as the Internet, power grid and traffic flow. In nature, complex network systems are evident in social interactions, where "Everybody on this planet is separated by only six other people. Six degrees of freedom. Between us and everybody else in this planet" [1]. In the biological world, one can see this complex interconnectivity in networks describing protein-protein interactions, metabolic pathways, transcriptional regulation, and neural connectivity.

### 37.1.2 Historical Perspective and Mathematical Preliminaries of Networks

The first study of networks, in the form of graph theory, comes from the Königsberg problem solved by Euler in 1736 [1]. Real networks are also random graphs as reported by Erdős and Rényi in 1959 [2], and this ER (Erdős-Rényi) view of networks has dominated scientific thinking for the past 50 years. Networks are highly interconnected. This was first proven in a social setting by Miligram in 1967, with people in the world being connected by a chain of six "friends of a friend" and leading to the phrase "six degrees of separation" between human beings, implying "it's a small world!" [3]. The concept of the small-world networks was revisited by Watts [4] with the availability of high-performance computing.

The basic mathematical concept used to model biological networks is a graph. A graph is a collection of vertices/nodes and interconnecting edges/links (Figure 37.1).

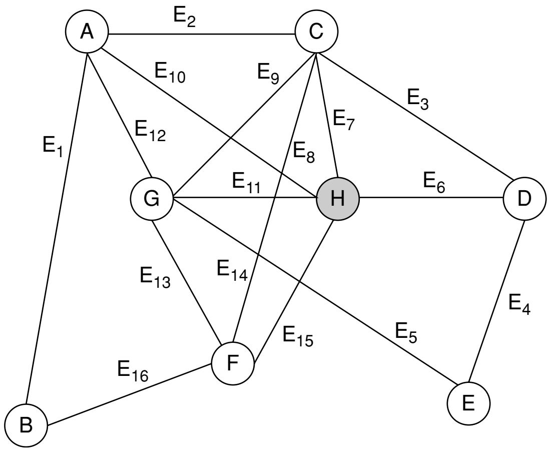


**Figure 37.1** Schematic representation of a graph. Nodes or vertices are represented in circles (*A*, *B*, . . . , *H*), whereas edges or links are shown by lines ($E_1$,$E_2$, ... , $E_{16}$). Highly connected nodes are known as *hubs* (*e.g.*, the gray node *H*).

In the context of cellular mapping, one can define nodes as proteins, metabolites, or genes. An edge is a connection between cellular nodes defining their functional or physical interaction. Understanding how the complex rules of molecular associations give rise to precise cellular and developmental processes has become a major challenge for modern molecular and cellular biology. A detailed review of the applications of graph theory in bioinformatics is presented in Chapter 8 [5].

A graph or network that we encounter in biological systems can be divided into two broad classes: *directed graphs* and *undirected graphs*. Directed graphs represent networks with a unidirectional interaction or connectivity between the nodes, whereas there is no such restriction in an undirected graph or network. When the first or initial node of a network is also connected to the final or last node, we have a cyclic graph. In an acyclic graph, the initial nodes are not connected to the final nodes, leading to an open network. Figure 37.1 represents an undirected graph, with 8 nodes ($A$, $B$, ..., $H$) and 16 edges ($E_1, E_2, \ldots, E_{16}$). As A is connected to the last node $H$, this graph is also cyclic.

In biology, transcription factor binding and metabolic networks are usually modeled as directed graphs. For example, in a transcriptional network, nodes would represent genes with edges denoting the interactions between them. For instance, if $X$ regulates $Y$, then there is a natural direction associated with the edge between the corresponding nodes $X$ and $Y$, starting from $X$ and ending at $Y$. This is also true of neuronal networks, where individual neurons represent nodes and synaptic connection corresponds to edges. Formally, a *directed graph G* consists of a set of *vertices/nodes* $V(G)$,

$$V(G) = \{v_1, v_2, v_3, \ldots\ldots, v_n\}$$

together with an *edge* set, $E(G) \subseteq V(G)xV(G)$. Intuitively, each *edge* $(u, v) \in E(G)$ can be visualized as connecting the starting node $u$ to the terminal node $v$. By convention, $uv$ is denoted as a short form for the edge $(u, v)$. It implies that the edge $uv$ starts at $u$ and terminates at $v$.

An *undirected graph G* consists of *vertex* set $V(G)$ and an *edge* set $E(G)$ with no direction associated with the edges. Hence, the elements of $E(G)$ are simply two elements of the subset of $V(G)$, rather than an ordered pair present in a directed graph, and can be represented as $uv$ or $vu$, to denote connectivity from one node to the next. For two vertices, $u$ and $v$ of an undirected graph, $uv$ is an edge if and only if $vu$ is also an edge. Among biological networks, protein-protein interactions are usually modeled as undirected graphs. For an undirected graph $G$ and a vertex $u \in V(G)$, the set of all *neighbors* of $u$ is denoted as $N(u)$ and is given by

$$N(u) = \{u \in V(G) : uv \in E(G)\}$$

The number of vertices $n$ in a directed or undirected graph is the *size* or *order* of the graph. For an undirected graph $G$, the degree of a node $u$ is simply the total number of edges meeting at $u$. In Figure 37.1, node $C$ has a degree of 4. The sum of the degrees of vertices in a graph equals twice the number of edges.

For a graph $G = (V, E)$,

$$\sum_{v \in V} \deg(v) = 2|E|$$

In an directed graph $G$, the *in-degree*, $deg_{\text{in}}(u)$ of a vertex $u$ is given by the number of edges terminating at $u$. On the other hand, $deg_{\text{out}}(u)$ (the *out-degree*) of a vertex $u$ is defined as the number of edges starting from the vertex $u$. Suppose that the vertices of a graph (directed or undirected) $G$ are ordered as $v_1, v_2, \ldots\ldots, v_n$. Then the adjacency matrix $A$ of $G$ is given by

$$\mathrm{A} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \qquad a_{ij} = \begin{cases} 1 & \text{if } v_i v_j \in E(G) \\ 0 & \text{if } v_i v_j \notin E(G) \end{cases}$$

The first example of a complex network that has attracted enormous attention is the *World Wide Web* (www). www consists of web pages as nodes and may be viewed as a directed graph when the first web page is linked to the second and so on. While studying the topology of the web, Barabasi and his group [6] showed that the in-degree has a power-law exponent of 2.1 and the out-degree has an exponent of 2.45, where power law is described as the probability $P(k)$ of finding the node with degree $k$ and is proportional to $k^{-\lambda}$. A similar power-law distribution was observed in a biological system for the first time by Liljeros *et al.* [7] while investigating sexual networks in Sweden. They found that the exponential degrees ($\lambda$) for female and male contacts are 2.5 and 2.3, respectively. A power-law degree distribution implies that a real network has no characteristic or central node. Although there are few nodes with a large number of neighboring nodes, the majority of nodes have only a few neighbors. The power-law degree distribution thus forces us to abandon the idea of a scale or a characteristic node. There is thus no scale in these networks. Hence, the term *scale-free* network was coined by Barabasi [8-10] from this observation of the power-law degree distribution (Figure 37.2).

The "fat-tail" of power-law distribution, in which there are few nodes with high degrees, consists of highly connected nodes called *hubs* [11, 12]. This kind of scale-free network is encountered with protein-protein interaction data, where the presence of hub proteins provides robustness to the system, as these hubs are also conserved more than non-hub proteins [12] across species. Networks depicting metabolic pathways [13, 14], protein domains [15], protein-protein interaction [16], regulatory genes [17], and transcription factor binding [18] interactions are all characteristically scale free.

### 37.1.3 Structural Properties of Biological Networks

The network structure plays a very important role in understanding the architecture and robustness of biological networks. Several of the commonly used

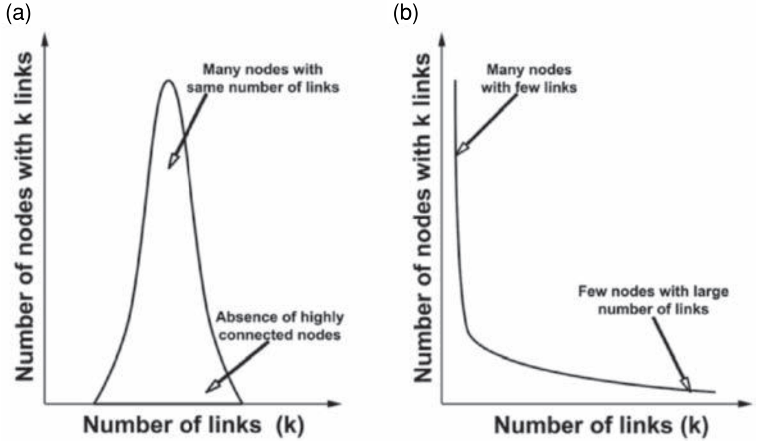


**Figure 37.2** Degree distribution of random network follows a bell-shaped curve (*i.e.*, normal distribution) (a), suggesting that most nodes have the same number of links, whereas the power-law degree distribution of a scale-free network shows the presence of many nodes with a few links. These nodes are held together by a few highly connected hubs (b).

topological features include degree distribution, short path length, clustering coefficient, and centrality measures. There are at least three structurally different classes of networks based on their degree distribution [19]:

1. *Single-scale* networks where the degree distribution has a fast decaying tail, such as exponential or Gaussian and is “small world” as defined by Watts and Strogatz [21].
2. *Scale-free* networks with power-law degree distribution $P(k) \sim k^{-\lambda}$ (described above).
3. *Broad-scale* or truncated scale-free networks for which $P(k)$ has a power-law regime followed by a sharp cutoff (*e.g.*, exponential or Gaussian decay of the tail). An example is the movie-actor network described in the small world of Watts [22, 23].

To characterize small-world networks, a measure of the shortest path lengths within the network is required. For a graph $G$ with $u$ and $v$ as two vertices, the path from $u$ to $v$ will pass sequentially through vertices $v_1, v_2 \ldots v_k$, with $u = v_1$ and $v = v_k$, such that for $i = 1, 2, \ldots\ldots k - 1$: (i) $v_i v_{i+1} \in E(G)$ and (ii) $v_i \neq v_j$ for $i \neq j$. The path length is then said to be $(k - 1)$. The simple *geodesic distance*, $d(u, v)$, from $u$ to $v$ is the length of the shortest path from $u$ to $v$ in the graph $G$. One can apply Floyd’s algorithm [24] to find all possible shortest paths in a network proportional to $V^3$, where $V$ is the total number of nodes or vertices in the graph $G$. The distance is infinity if no such path exists (*i.e.*, $d(u, v) = \infty$). In a connected graph $G$, every possible vertex is connected to another (*i.e.*, $(u, v) \in V(G)$). The average path length, $\langle l \rangle$, of such a graph is defined as the average of values taken over all the possible pairs of nodes connected by at least one path:

$$\langle l \rangle = \frac{2}{N(N-1)} \sum_{i=1}^{N} l_{ij}$$

The diameter of the network is defined as the maximum distance between two nodes of a graph $G$ (*i.e.*, $D = \max\{d_{ij} | i, j \in N\}$, where $N$ is the total number of nodes in the graph or network). The average network diameter is given by

$$\langle D \rangle = \frac{\sum_{N} N f(N)}{\sum_{N} f(N)}$$

where $f(N)$ represents the frequency of the shortest path length between two nodes. The diameter $D$ is a global property whereas the average diameter $\langle D \rangle$ is a local property of the biological network. The characteristic path length and the diameter in a network are indicators of how readily "information" can be transmitted through the network. This small-world feature is observed in many biological networks, suggesting its efficiency in information transfer (*i.e.*, only a small number of intermediate steps is necessary for any one protein, metabolite, or gene to influence the characteristics or behavior of another biomolecule) in a complex network.

The degree distribution is a local characterization of a network and can be meaningfully interpreted when the network is based on an ER model [2, 20] or a Barabasi-Albert (BA) model [8, 9]. The clustering coefficient is another characteristic of a network that is unrelated to the degree distribution. It is a quantitative measure to the proximity of the neighborhood of each node to form a complete subgraph (clique) [21] and thus defines a measure of the local behavior of the small-world network. The clustering coefficient is defined as

$$C_i = \frac{2N}{k_i(k_i - 1)}$$

where $N$ denotes the number of existing links among the $k_i$ nodes connected to the node $i$. Similarly, one can define an average clustering coefficient as

$$\langle C \rangle = \frac{1}{N} \sum_{i=1}^{N} C_i$$

An additional measure of the network structure is the function $C(k)$, defined as the average clustering coefficient of all nodes with $k$ links. $C(k)$ may be dependent or independent of $k$. If $C(k)$ is independent of $k$, then the network has a homogenous representation of many small, tightly linked clusteres. Otherwise, if the function takes the form $C(k) \sim k^{-1}$, that network has a hierarchical representation. Thus, as the degree of a node increases, its clustering coefficient $C(k)$ decreases, implying that the neighborhoods of low-degree nodes are densely clustered, whereas those of hubs are sparsely clustered. Ravasz *et al.* [25] calculated the average clustering coefficient in a metabolic network of 43 organisms and reported that $C(k)$ of the metabolic network was at least an order of magnitude higher than that of the corresponding BA network. Similar observations were reported for the interacting pairs of yeast nuclear proteins, comprising 318 interactions among 329 proteins, where most of the neighboring nodes of hub proteins have significantly lower connectivity than the hubs [26]. Dense local clustering is exhibited in the yeast genetic interaction network, consisting of ~1000 genes with ~4000 interactions [27].

Centrality is one of the key structural aspects of the nodes in a network and is a measure of the relative influence of each node on the network. It thus ranks the nodes based on the extent of connections around each node. Specifically, centrality defines the share of the total centrality ascribed to the most central node. In biological networks, the centrality measure helps in identifying the important genes or proteins in the network. For example, in the noncancerous cell, *p*53 gene is inactive, whereas in a tumor cell the *p*53 network is activated, resulting in the stimulation of enzymes that modify *p*53 and its negative regulator MDM2 [28]. Jeong *et al.* [12] showed that in the yeast protein interaction network, the highly connected proteins are three times more essential compared to proteins with only a few connections, highlighting the importance of centrality.

There are four classical variants of centrality measures in network theory:

1. Betweenness centrality
2. Closeness centrality
3. Degree centrality
4. Eigenvector centrality

Detailed mathematical descriptions of the above are available from the excellent review of Mason *et al.* [29]. Betweenness centrality is the most common centrality measure in protein, gene, or transcription factor binding networks. It is the fraction of shortest paths between all the pairs of nodes that passes through a given node [30]. It gives the notion of traffic or flux through a given node, assuming that the information flow across a network primarily follows the shortest available path.

### 37.1.4 Local Topology of Biological Networks: Functional Motifs, Modules, and Communities

Beyond the properties of individual nodes (such as *hubs*) and pair nodes, one can observe structural organization that includes few nodes together. A recent study has shown the existence of a loose hierarchical structure within biological networks. At the lowest level, one can observe three to four node clusters with a significantly higher frequency relative to the randomized network. Such clusters are commonly referred as the *network motifs*. Milo *et al.* [31] developed a tool, *mfinder*, for the network motifs detection. The *mfinder* algorithm detects the network motifs by exhaustively enumerating all the subgraphs (clusters) within a given number of nodes in a network. When applied on the transcriptional regulatory network of *Sacchromyces cerevisae* (yeast or baker's yeast) and the neuronal network of *Caenorhabditis elegans*, he showed the presence of three-node *feedforward loops* (FFLs) and four-node *bi-parallel motifs*. A similar finding by Shen-Orr *et al.* [32] showed the presence of FFLs, *single-input motifs* (SIMs), and *dense overlapping regulons* (DORs), which are three common motifs present in the *Escherichia coli* transcriptional network. The core concept of network motif and basic schema is described in the method by Milo *et al.* [31]. The *mfinder* method can be described in three simple steps. For a given directed graph $G$, the motifs of size $k$ (three to seven nodes or vertices) can be identified as follows. First, calculate the frequency of the motif's occurrence in a directed graph $G$ (*i.e.*, the number of cluster $S$ (subgraph) of size $k$, in $G$). Second, generate a large number of random networks such that each node has the same in-degree and out-degree and every cluster of size $k - 1$ occurs with the same frequency as in the real network $G$. Third, one can describe cluster $S$ as a motif if it satisfies the following three conditions:

I. The probability of $S$ occurring in a real network is more frequent under a prescribed p-value compared with its corresponding random network.
II. There should be at least four distinct occurrences of $S$ in the real network $G$.
III. The actual number of occurrences of $S$ in $G$ is significantly higher than the average number of occurrences of $S$ in the randomly generated networks.

The above approach was used for the detection of network motifs. It was demonstrated that network motifs play a key informational processing role in many biological networks. Therefore, it is worth mentioning the general structural features (Figure 37.3):

i. The FFL motif describes a situation where two transcription factors A and B jointly regulate a third transcription factor C, and A also regulates B positively (coherent FFL) or negatively (incoherent FFL) [33]. This can be seen in the L-arabinose utilization system [34]. These motifs occur where an external signal causes a rapid response in the biological system. The presence of coherent FFL suggests a functional design where decision making is based on the fluctuation of external stimuli.
ii. SIM is described as a single transcription factor regulating many operons. It is generally autoregulatory and has a temporal design pattern for processing information that needs to be carried out in multiple steps. The classical example is arginine biosynthetic pathway. Many amino acid pathways contain such motifs [32].
iii. DOR is defined as the layer of overlapping interaction and a group of input transcription factors (TFs) that is much more dense corresponding to the randomized network [32].

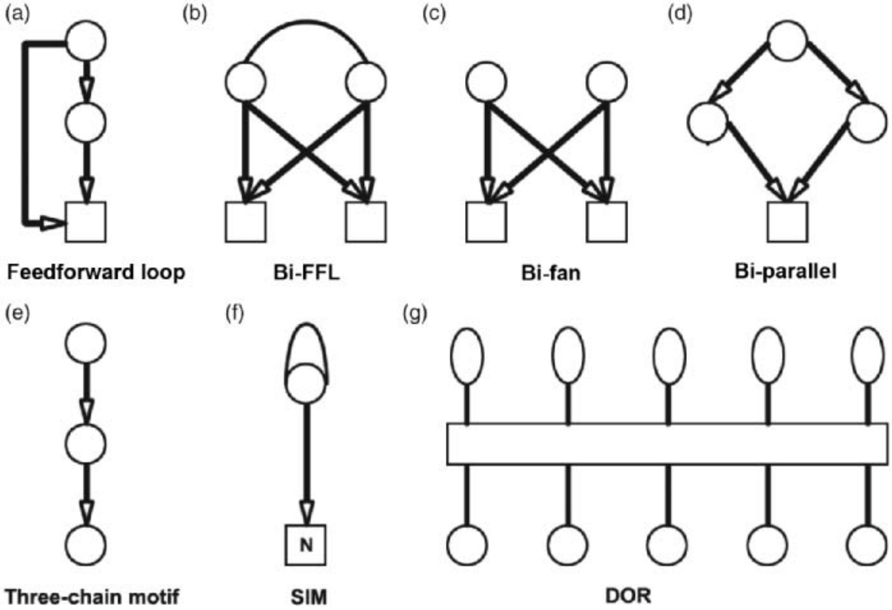


**Figure 37.3** The common motifs with three-unit and four-unit subgraphs commonly present in biological networks. TFs or regulators are marked with a circular/elliptical object, and the target gene is represented as a box. (a) FFLs, in which one regulator controls the other and both of them together control the target gene. (b) Bi-FFL motifs, one TF controls another and both of them bind to common target genes. (c) Bi-fan motifs, where two separate regulators bind together with two target genes. (d) Bi-parallel motifs, in which one TF controls two separate regulators that further regulates a given target gene. (e) Three-chain motifs, where three regulators (in this case, it need not be the TF) are controlled by each other in a stepwise manner for controlling a given biological process. (f) SIM, a single TF controls a set of operons and the TF usually is autoregulatory. (G) DOR, defined as a set of genes, controlled by the set of TFs.

iv. The bi-fan motif is the simple version of the DOR motif in which two transcription regulators bind to a common target. This suggests a simple functional design to precisely control the large numbers of targets under several different conditions by a few regulators. This four-node motif is common in the *Caenorhabditis elegans* neuronal network [32].

v. The bi-parallel motif is found in transcriptional and signaling networks of many organisms and indicates redundancy. It is a four-node motif that comprises a regulator (transcription factor) controlling two other regulators that further regulates one gene. This motif has a wide range of functional roles in biological networks [31].

vi. Three-chain motifs are frequently present in food web networks. This motif represents structural adaptation to the energy flow in the food web were flow is directed from the lower tropic level to the higher tropic level in the food pyramid [31].

**Table 37.1 Biological networks visualization tools**

| | Purpose | Website |
|---|---|---|
| *Biolayout* | Visualization; analysis | http://biolayout.org/ |
| *Cell Illustrator* | Visualization and analysis | http://biobase-international.com |
| *Cytoscape* | Visualization; analysis | http://cytoscape.org |
| *Graphviz* | Visualization; analysis | http://graphviz.org |
| *H3Viewer* | Visualization; analysis | http://graphics.stanford.edu/~munzner/h3/ |
| *LaNet-vi* | Online visualization tool | http://xavier.informatics.indiana.edu/lanet-vi/ |
| *NetMiner* | Visualization and analysis | http://netminer.com/NetMiner/home_01.jsp |
| *Osprey* | Visualization; analysis | http://biodata.mshri.on.ca/osprey/servlet/Index |
| *Pajek* | Visualization; analysis | http://vlado.fmf.uni-lj.si/pub/networks/pajek/ |
| *Visant* | Visualization; analysis | http://visant.bu.edu/ |
| *Yed* | Graph editor | http:/yworks.com/en/products_yed_about.html |

The concept of network motifs detection is further extended based on the random sampling of subgraph for estimating subgraph concentration [35]. The significance profile (SP) descriptor was proposed as a mean for classifying networks based on subgraphs [36]. FANMOD is another useful and fast network motif detection tool in bioinformatics, which improves the motif detection by removing the sampling bias in a random network and scales well with varying node degrees in a network [37–39]. Tables 37.1 and 37.2 provide additional information on bioinformatics network motifs visualization and analysis tools.

Most real-world networks contain parts/subsets in which group of nodes are highly connected to each other rather than to the rest of the network. Such sets of nodes are usually referred as cluster, communities, cohesive groups, or modules [39–41]. Ravasz *et al.* [25] observed the presence of modules or communities in the metabolic network, suggesting a hierarchical organization of the biological network. Hierarchy describes the structural organization in a network (*i.e.*, how nodes

**Table 37.2 Biological networks analysis tools**

| | Purpose | Website |
|---|---|---|
| *Bioconductor* | Network analysis | http://bioconductor.org/ |
| *Biotapestry* | Network analysis | http://biotapestry.org/ |
| *Cfinder* | Visualizing dense cluster | http://cfinder.org/ |
| FANMod | Network motif detection tool | http://theinf1.informatik.uni-jena.de/~wernicke/motifs/ |
| MAVisto | Motif analysis and visualization | http://mavisto.ipk-gatersleben.de/ |
| *Mfinder* | Network motif detection tool | http://weizmann.ac.il/mcb/UriAlon/groupNetworkMotifSW.html |
| TYNA | Network analysis | http://tyna.gersteinlab.org/tyna/ |
| *Vanted* | Network analysis | http://vanted.ipk-gatersleben.de/ |

in a network link to form a motif). These motifs combine to form communities, and these communities link to form a network. Communities occur between the scale of the whole network and the scale of the motifs. Girvan and Newman [42] proposed an algorithm for the community detection around the ideas of centrality indices (as mentioned above). They introduce an *edge betweenness*, which is defined as the number of the shortest paths between pairs of vertices that run along it. In order to detect communities inside a network, one should look for all the possible short paths between few edges, which loosely connect the local community in a network. Their algorithm has a run time in the order of $O(n^3)$, where $n$ is the number of nodes. Newman defines *modularity* as a property that describes the division of a network into a distinct community [43]. Communities are characterized by having more links within them (between constituent nodes) than with other communities (between the constituent and nonconstituent nodes). Communities are determined using the measure

$$Q = \frac{1}{L}\sum_{i=1}^{n}\left(li - \frac{di^2}{4L}\right)$$

where $Q$ is the measure of modularity to define the partition. $L$ denotes the total number of links in a network with $n$ communities, $l_i$ denotes the number of links in the $i$th community, and $d_i$ is the total number of degrees in the community $i$. Clauset *et al.* [44] proposed the modularity optimization algorithm to enhance the run time for finding the communities inside a network. Many real-world networks are sparse (no. of edges $\sim$ no. of vertices) and hierarchical (depth $d$ of the dendrogram such that $d \sim \log n$, where $n$ is the number of vertices in a network). Their modularity optimization algorithm for most of the real network runs in linear time of the order $O(n\log^2 n)$. The *clique percolation method* (CPM) algorithm was developed to quantify the overlapping community [41]. It is a promising algorithm for predicting the functional relationship of proteins in an interaction network. *CFinder* is a free-software tool that implements the CPM algorithm (Table 37.2). To gain an insight into the metabolic network of *Treponema pallidum,* Clauset *et al.* [45] proposed the hierarchical random graph model. The ability of this algorithm to detect false-positive and missing links in a complex network makes it valuable for deciphering the hidden structure of the biological system.

A revolutionary approach to discovering gene function has been to knock out a gene and observe its phenotype. One such computational method is based on the framework of*flux balance analysis* (FBA). Segre *et al.* [46] studied the system-level epistasis interaction by computing the phenotype of all single and double knockout of 890 metabolic genes in *S. cerevisae*. FBA is a mathematical approach for computing whole-cell metabolic fluxes and the growth rate on the steady-state and optimality assumptions. Based on their finding, they developed the *Prism* algorithm for hierarchical clustering of gene pairs for the observed epistasic behavior in the yeast cell. The promising computational algorithms discussed above can be very helpful in identifying the function of genes and proteins in the biological network.

## 37.2 TYPES OF BIOLOGICAL NETWORKS

### 37.2.1 Protein-Protein Interaction Networks

To date, protein-protein interaction maps represent the largest and the most diverse dataset available on the biomolecular networks. The first protein interaction map was developed using the yeast two-hybrid technique [47, 48]. This technique has also been used to study the protein interaction map of species such as *Drosophila melanogaster, C. elegans*, and *Homo sapiens* [49–51]. The recent use of high-throughput screening techniques using affinity purification followed by identification of associated proteins using mass spectroscopy has resulted in large datasets of protein interactions.

The use of protein interactions to predict function relies on the principle assumption that the interacting protein pairs are likely to collaborate for a common purpose. Schwikowski *et al.* [16] was the first to show that the *S. cerevisiae* interaction network (containing 2358 interactions among 1548 proteins) could be used to classify protein into functional groups simply by observing the function of the neighboring proteins. Their approach correctly classified 63% and 76% of interacting protein pairs into the same function and the same subcellular location, respectively. The idea of "guilt by association or neighboring count method" was extended by providing a $\chi^2$-like functional score for the protein functional assignment [52]. This functional score accounts for the influence of distant neighbors when predicting the function of a target protein in a given protein interaction network. For a target protein, the highest $\chi^2$ value among the function of all $n$-neighboring proteins can be defined as

$$\chi_i^2 = \frac{(n_i - e_i)^2}{e_i}$$

where $i$ denotes the protein function, $e_i$ denotes the expected number of $i$ in $n$-neighboring proteins expected from the distribution of proteins in the whole network, and $n_i$ denotes the observed number of $i$ in $n$-neighboring proteins. One problem with this method is that it treats distant as well as nearest neighbors for a given protein equally (*i.e.*, the functional score does not take the distance into account during the functional assignment of protein). To overcome this problem, a functional score that gives a different weight to neighboring proteins according to their distance from the target protein was proposed [53]. This method gives importance to the functional similarity of proteins that are close in the interaction network.

The majority rule method predicts the function of a target protein to be the best represented function among the neighbors of known function. Predicting the function of an unknown protein when its neighbors include more proteins of an unknown than a known function can be difficult using the simple majority rule. To address this scenario, the simulated annealing method for the functional prediction of protein was proposed [54]. The simulated annealing method takes a generic approach to account for the fact that yeast two-hybrid data itself contains false-positive and false-negative information about the interaction in the protein network map. In this method, proteins of unknown function are randomly labeled with a function. A score $E_x$ is calculated based on the observed functional similarity of neighboring proteins in the network. A protein of unknown function is then randomly relabeled with another function, and the corresponding score is calculated ($E_y$). If $\Delta E$ ($E_y - E_x$) $\leq 0$, then the new functional labeling (with score $E_y$) is retained. Otherwise, the new function is accepted with a probability $r = \exp(-\Delta E/T)$, where $T$ represents a hypothetical parameter. Nabieva *et al.* [55] suggested a graph-based network-flow algorithm to predict the protein function in an interaction network. This approach accounts for the decreased likelihood of proteins having the same function at greater distances. The basic idea is to treat each protein annotated with the function as the source of a "functional flow." After simulating the spread of function over time through the network, each unannotated protein is assigned a score for each function based on the amount of that function that "flowed" to the protein during the simulation.

One can also compare protein interaction networks to identify *interologues* (interactions that are conserved across species). Algorithms such as *PathBlast* and *NetworkBlast* were proposed to identify paths and a dense cluster of conserved interactions [56, 57]. These methods are important for comparing the cellular machinery of different species and for increasing the overall confidence in the underlying interaction measurements.

### 37.2.2 Metabolic Networks

All living entities, whether unicellular or multicellular organisms, consist of similar chemical molecules for extracting and using energy from the surrounding environment. This process is known as metabolism. Directly, it refers to the thousands of chemical reactions and molecules that govern the flow of mass and energy in an organism. The sequential nature of a chemical reaction in a metabolic pathway means that it can be modeled in the form of a graph or network. The metabolic network usually focuses on the flux (steady-state mass flow) in basic biochemical pathways that generate essential components such as polymers (proteins, polysaccharides, lipids, and polynucleiotides) and small molecules that are the basic building block of polymers. Literature curation and genome annotation were used to construct these complex metabolic networks in organisms such as *E. coli*, yeast, and humans (Table 37.3). One can describe the metabolic pathway or network as a collection of connected biochemical molecules. As such, these networks are often represented as directed graphs, typically containing both metabolites as well as proteins (*i.e.*, enzymes). Metabolic pathways can be "metabolite-centric" or "enzyme-centric." The metabolite-centric metabolic network is the most common representation of metabolic pathways (Figure 37.4).

Wagner and Fell [14], in their undirected graph analysis of an *E. coli* metabolite network, showed that the presence of a highly connected metabolite like coenzyme *A*, NAD, ATP, or GTP gave an evolutionary glimpse of the RNA world. Their metabolite-centric network analysis also suggests that the glycolysis and TCA cycles are the most ancient metabolic pathways. The intermediate metabolites of Morowitz [58] such as pyruvate, 2-oxoglutarate, acetyl coA, and oxalate are the core

**Table 37.3 Interaction and regulatory databases**

| | Purpose | Website |
|---|---|---|
| *BioCyc* | Metabolic pathway | http://biocyc.org/ |
| *BioGrid* | Metabolic pathway | http://www.thebiogrid.org/ |
| BOND | Metabolic pathway | http://bond.unleashedinformatics.com/Action? |
| DIP | Protein-protein interaction | http://dip.doe-mbi.ucla.edu/dip/Main.cgi |
| HPRD | Protein-protein interaction | http://www.hprd.org/ |
| *IntAct* | Protein-protein interaction | http://www.ebi.ac.uk/intact/site/index.jsf |
| JASPER | Transcription factor binding profile | http://jasper.cgb.ki.se/ |
| KEGG | Encyclopedia of gene and genome | http://www.genome.jp/kegg/ |
| MINT | Protein-protein interaction | http://mint.bio.uniroma2.it/mint/Welcome.do |
| MIPS | Protein-protein interaction | http://mips.gsf.de/ |
| *String* | Protein-protein interaction | http://string.embl.de/ |
| TRANSFAC | Transcription factors | http://www.biobase-international.com/pages/index.php?id=transfac |

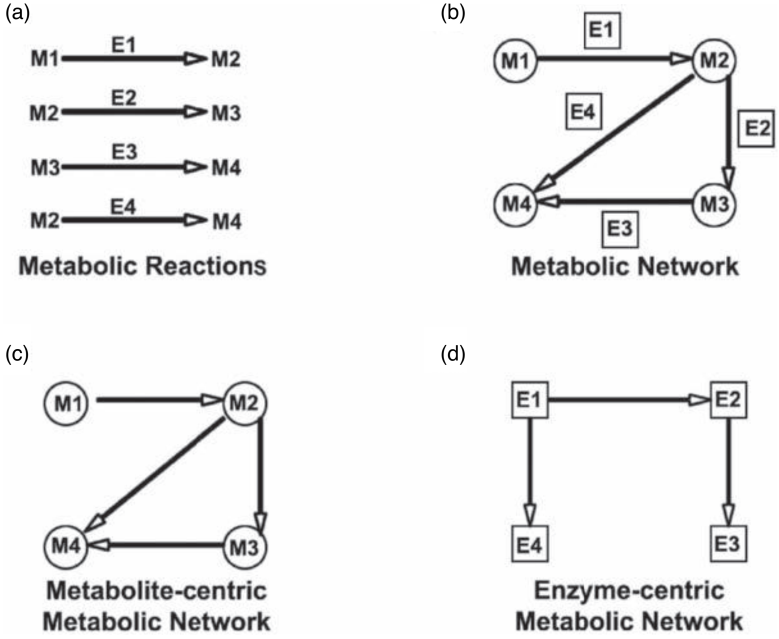


**Figure 37.4** Schematic representation of metabolic networks. (a) A set of metabolic reactions involving the interconversion of metabolites M1, M2, M3, and M4, catalyzed by enzymes *E*1, *E*2, *E*3, and *E*4. (b) A metabolic network representation of the set of reactions in (A). (c) A metabolite-centric metabolic network showing the connectivity among the metabolites in (A). (d) An enzyme-centric metabolic network showing the connectivity between the enzymes in (A).

metabolites in the ancient pathway of the molecular evolution. Wagner and Fell's study implied that metabolic pathways evolve by preferential arrangements, which lead to small-world architecture. This topology of the metabolic network helps in fast response to any perturbation posed by the surrounding environment. In many obligate endocellular symbionts, biochemical pathways comprise intermediate metabolites as a minimal set [59]. A comprehensive study of metabolic networks from more than 300 bacterial species (taken from the KEGG database) suggested that network size is an important topological determinant of modularity in the metabolic network [60]. Larger genomes have a high modularity score compared with smaller genomes. Environmental factors also contribute to modularity across different species. Pal *et al.* [61] showed that metabolic networks evolve *via* horizontal gene transfer which contributes to the fitness for its specific environment. They showed that horizontally transferred genes are integrated at the periphery of the network, whereas the central network pathways (*e.g.*, TCA cycle and glycolysis) remain evolutionarily stable.

Interactions between metabolites and enzymes in metabolic networks are closely related to their gene functions. This suggests the potential role of genes in shaping the network behavior and its wider application to achieve the desired metabolic products. In view of this, considerable attention has been focused on network dynamics using constrain-based analysis such as the FBA. FBA assumes a steady state of all metabolites and that living organisms have optimized metabolic fluxes for maximizing biomass production [62–64]. The *OptKnock* framework was suggested to incorporate this key concept, which uses the gene deletion strategies to overproduce a specific chemical compound in *E. coli* [65]. This method was applied to the production of succinate, lactate, and 1, 3-propanediol (PDO) in *E. coli.*

Environmental conditions play a major role in the viability of a cell. The Papp *et al.* [66] study suggests that the majority of nonessential enzymes in the yeast metabolic network are important for the viability of cells under certain environmental conditions. In their study, only a small subset of network flux is compensated for the presence of enzyme or the alternative biochemical pathways. Highly connected enzymes in the biochemical pathway of yeast have direct access to many network nodes that resulted in the short path length in the metabolic network [67]. The metabolic network suggests that the highly connected enzymes of large flux have undergone less mutation. The network model suggests the lowering of enzymatic flux caused by mutation. However, high flux enzymes have evolved through the gene duplication event. These features of metabolic networks should be general to other organisms as fundamental metabolic network structures are likely to be conserved during evolution. Indeed, a topological analysis of metabolic networks in 43 organisms from all three domains of life (eukaryote, archaea, and bacteria) revealed the presence of highly similar topological properties [13].

### 37.2.3 Transcriptional Networks

The genetic program of any living organism is encoded in its genome. The functional unit of an organism's genome is the gene. Gene expression programs involve the regulation of thousands of genes. TFs are an essential subset of interacting proteins responsible for controlling the expression of genes. The pioneering studies by Jacob and Monod suggested the presence of TFs, which interact with DNA elements upstream of specific genes and control their expression, by forming transcriptional regulatory complexes. Several studies during the past few years have accumulated a wealth of information about the transcriptional regulatory complexes in model organisms [68–71]. The complexity of an organism seems to be linked to their number of TFs. Organisms such as *E. coli* and *S. cerevisiae* seem to have 200–300 TFs. Compartive analysis of multicellular eukaryotes shows the presence of 600–820 TFs in *C. elegans* and *D. melanogaster*, and 1500–1800 in *Arabidopsis thaliana*. Around 1500 TFs have been documented for humans, although an estimate of 2000–3000 TFs is predicted [72]. Phylogenetic analysis has revealed that the growth of certain transcription factor families relies on the amplification and shuffling of protein domains.

Transcription factor-binding networks have been assembled in two different ways.

I. Through large-scale, genome-wide location analysis of the cis-regulatory (promoter) region of DNA. The sea urchin embryo gene regulatory network was developed in this manner, controlling the specification of the endoderm and mesoderm tissue formation [17].

II. Through large-scale identification of transcription factor-binding sites using chromatin immuno-precipitation followed by probing of genomic microarrays (ChIP-chip) or DNA sequencing (ChIP-PET or STAGE). This method has been used for creating the transcriptional network in yeast and other higher organisms [68–74].

Location analysis for identifying interaction between regulators (TFs) and DNA regions provides strong evidence of transcriptional regulation. Although this information is useful, it is limited because TF binding does not provide any functional evidence in terms of regulation (*i.e.*, whether a positive, negative, or silent regulator). On the other hand, microarray-based location information contains substantial experimental noise. Since both sets of information are useful and complementary to each other, Bar-Joseph *et al.* [75] developed the GRAM (*Genetic regulatory module*) algorithm to integrate these data sources. A similar approach to integrate genomic data for yeast has been developed in the SANDY algorithm [18] and applied to the *S. cerevisiae* transcriptional network. The algorithm was tested on a yeast transcriptional regulatory network under five different experimental conditions to capture the network dynamic. It was found that 78% of the hubs are transient and seem to influence one experimental condition and not the others, suggesting a relationship to condition-dependent lethality. Lee *et al.* [76] showed that the yeast transcriptional network can rewire itself to describe potential pathways for global gene regulation and expression programs. This study also showed that 10% of yeast TFs is autoregulated, whereas a prokaryote (*E. coli*) was shown to have 52–72% TFs under autoregulation. The transcriptional network of yeast shows the presence of 188 regulator chain motifs, consisting of 3–10 TFs. These motifs are present mainly in the regulatory circuit of the cell cycle, associated with the regulation of the different stages of the cycle. The presence of *multiple component loop* (MCL) motifs seems to be characteristic of eukaryotic transcriptional networks, as no such motif is found in prokaryotes.

Several experimental and computational studies suggest that combinatorial regulation is the dominant mechanism behind complexity in gene expression patterns. To understand how gene expression is controlled by TF binding and thereby attains functional co-regulation through combinatorial association, Balaji *et al.* [77] studied the co-regulation network using the yeast transcriptional network. Their network is defined by adding an edge between a pair of TFs denoting a co-regulation association, when a pair of TFs regulates the same target gene. Their results suggest that 67% of the TFs seem to have a positive correlation, in a co-regulation network consisting of 5622 pair-wise associations between 157 TFs.

The *human transcription factor network* (HTFN) seems to be dissociative, with higher degree proteins attached to many low-degree ones [72]. This is an important property as it signifies the presence of modularity and its functional importance, where a smaller set of TFs closely regulate a group of genes. A dissociative network is also characterized by hubs that are linked to other network elements but not usually to each other. This dissociativeness allows large parts of the network to be separated and thus partially isolated from different sources of perturbation. Experimental studies suggest that many TFs binding sites lack biological function, or, more likely, are functionally redundant with other regulatory sites or affect gene expression under other conditions. In mammalian gene expression systems, TFs bind to distant locations on genes, rather than to upstream locations.

### 37.2.4 Other Biological Networks

*In vivo* cellular studies of human and yeast proteins suggest that 30% of proteins are phosphorylated [78]. Signaling pathways recruit protein kinases, which are capable of undergoing phosphorylation for initiating cellular cross-talk, that trigger subsequent biological processes. Weng *et al.* [79] used a neural network simulator, GENESIS, to analyze a simplified network consisting of four different sections of the MAP kinase pathways. Plants also use a similar kind of signaling pathways for physiological processes. *Abscisic acid* (ABA) signal transduction in the guard cells is one of the best characterized signaling systems in plants. The ABA signaling pathway contains more than 20 components, including signaling proteins, secondary metabolites, and ion channels. ABA induces guard cell shrinkage and stomatal closure *via* secondary messengers such as calcium ion and cytosolic pH. Dynamical modeling of the guard cell ABA network by Li *et al.* [80] showed that gene disruptions and pharmacological interventions do not affect a significant fraction of the network, thereby validating its robustness. This network analysis also revealed a redundancy in signaling pathways as eight independent pathways for ABA signaling. Many signaling proteins in this network are represented by multigene families in *Arabidopsis*, providing functional redundancy as protein isoforms. Table 37.4 provides descriptions of the components of selected biological networks, across several organisms.

**Table 37.4 Current status of biological Networks**

| Types of Network | Species | Number of nodes | Number of Interactions | Reference |
|---|---|---|---|---|
| Protein-protein Interaction Network | *S. cerevisiae* | 1,548 | 2,358 | [16] |
| | | 1,825 | 2,238 | [47] |
| | | 1,415 | 2,135 | [48] |
| | *C. elegans* | 4,651 | 3,039 | [50] |
| | *D. melanogaster* | 1,307 | 2,483 | [49] |
| | *H. sapiens* | 7,509 | 20,979 | [99] |
| Transcriptional Regulatory Network | *S. cerevisiae* | 2,343 (genes) | ~4,000 | [78] |
| | | 142 (TFs) & 3,420 | 7,040 | [20] |
| | | (genes) | 12,873 | [79] |
| | | 157 (TFs) & 4,410 | — | [74] |
| | | (genes) | — | [18] |
| | *H. sapiens* | 243 (TFs) | | |
| | *Sea Urchin* | 40 (genes) | | |
| Metabolic Network | *E. coli* | 283 (Metabolite) | — | [14] |
| | *S. cerevisiae* | 1,881 (enzymes) | — | [66] |
| | *M. genitalium* | 83 (enzymes) | — | [67] |
| Genetic Network | *S. cerevisiae* | 3,258 | 13,963 | [99] |

## 37.3 NETWORK DYNAMIC, EVOLUTION AND DISEASE

### 37.3.1 Biological Network Dynamic and Evolution

A generic property of the complex network is that it constantly evolves in time. This implies that the underlying networks are not static but change continuously through the addition and/or removal of new nodes and links. Barabasi and Albert [8] investigated the WWW and proposed a preferential attachment model for the growth of this scale-free network. The BA model was not inspired by any specific biological consideration. It is a mathematical model for the dynamical growth of any scale-free network that captures the consequence of two generic mechanisms:

1. *Growth*: a network grows by adding new nodes to it.
2. *Preferential attachment*: a new node added preferentially to those nodes that are already well connected.

Eisenberg *et al.* [81] supported this growth model for the evolution of a protein interaction network by cross-genome comparisons. They used a BLAST e-value as a sequence similarity measure to classify yeast proteins into a different evolutionary time-scale by genome-wide comparisons with *E. coli*, *Arabidopsis*, and fission yeast. The assumption made in this study is that a protein created at a certain time in a certain ancestor organism will have descendants in all organisms that diverge from this ancestor. An analysis of average connectivity of proteins in the yeast interaction dataset showed that there is a clear dependency of the connectivity on the age of the protein, with older proteins having significantly more interactions. To account for the preferential growth model, a comparison of the number of links for each protein in the oldest genome (*E. coli*) to the newer ones (fission yeast and *Arabidopsis*) showed that the average links per protein is 6.5 in the oldest genome (*E. coli*) and 0.5 in the newest (fission yeast). On a molecular level, one can expect such growth as there is a tendency to increase the number of protein attachment domains or improve existing domains such that they can bind to more target proteins. This mechanism of preferential attachment may depend on the physicochemical properties of the highly conserved and well-connected proteins. The network analysis supports the essentiality of hub proteins for the survival of the species as well as establishes their age on the evolutionary time-scale.

Bianconi *et al.* [82] introduce a fitness model to accommodate the fact that in many real networks, the connectivity and growth rate of a node do not depend on its age alone. The assumption made here was that the existence of fitness modifies the preferential attachment to compete for links. Proteins with more interactors evolve more slowly not because they are more important to the organism but because a greater proportion of the proteins are directly involved in function and evolve together as a unit [83]. Thus, proteins with many interactors have a large effect on the fitness of an organism.

Many recent models for complex network evolution are based on the mechanism of growth and preferential attachment. However, more biologically motivated models have been developed for protein interaction and genetic network. These models are based on the fundamental biological assumption of cellular processes such as "duplication" and "divergence." This hypothesis for the duplication-divergence model states that gene and protein networks evolve through the occasional copying of individual genes or protein coding genes followed by acquired random mutation that leads to functional diversification. Over a period of time, these processes can combine to yield a complex network of genes and protein interactions. Vazquez *et al.* [84] used their graph-generating models to mimic the evolutionary process of duplication and divergence of the genes that translate to proteins. We describe briefly the simulation of the duplication-divergence model of a protein interaction network, whose evolution is based on the following rules:

I. *Duplication:* Select node $x$ randomly and add an edge to node $x'$. The link is established between $x$ and $x'$ with a probability $P$.

II. *Divergence:* For each of the nodes $y$ linked to $x$ and $x'$, choose randomly one of the links $(x, y)$ or $(x', y)$ and remove it with probability $Q$, where $P$ is a graph-model parameter representing the creation of an interaction between the duplicates of a self-interacting. $Q$ represents the loss of an interaction between the duplicates and their neighbors due to the divergence of the duplicates. Based on these simple rules, Vazquez *et al.* showed the creation of a scale-free protein interaction network.

The mechanism of interaction gain and loss discussed above also shape the rewiring and cross-talk observed in metabolic pathways and transcriptional regulation. Luscombe *et al.* [20] showed the rewiring under “exogeneous states” (*e.g.*, stress response) leads to a decrease in the diameter of the transcriptional network in a yeast cell, with large hubs responding quickly to the external conditions and stimulus. Many signaling pathways often employ the same or homologous proteins for sensing external stimuli. For example, in *S. cerevisiae,* two pathways for sensing the osmolar and pheromone signals share the same or homologous molecules. Despite the potential cross-wiring through common molecules, cells show specificity of response by filtering out spurious cross-talk through mutual inhibition [85].

### 37.3.2 Biological Networks and Disease

Epidemic models can be mapped by the connectivity pattern characterizing the population in which an infective agent (*i.e.*, infection) spreads. For example, the web of sexual contact is the natural social network that defines the spread of *sexually transmitted disease* (STD) and its persistence. As most social networks are described by power-law degree distributions, the heterogeneity of connectivity patterns determines the epidemic outbreak in such scale-free networks. Pastor-Sattoras and Vespignani [86] first showed that the spread of infection is tremendously strengthened by using a *susceptible-infective-susceptible* model (SIS model) to analyze the underlying scale-free network. Their findings on this model suggest that the scale-free network supports the persistence of diseases, irrespective of any low values of infective rate. May and Lloyd [87] arrived at the same conclusions regarding the absence of any epidemic threshold, using a SIR (susceptible-infective-removed) model. The mathematical details of the SIS and SIR models are described by Hethcote [88]. The results of scale-free network analyses have resulted in orienting immunization strategies to target highly connected node individuals, in order to control disease propagation [89], because scale-free networks do not acquire global immunity from major epidemic outbreaks even in the presence of an unrealistically high density of randomly immunized individuals. These highly connected individuals are the hubs of social networks.

At the cellular level, disease is characterized by the malfunction of cellular processes resulting from absent/aberrant proteins. The protein coding gene can be of immense value to understand diseases as shown by linkage analysis in several infected families. There are diseases like PID (*primary immunodeficiency*) caused by defective endosomal adapator proteins p14, where no substantial statistical evidence is present for general understanding of the disease mechanism [90]. However if the disease-related locus alone is identified, subsequent gene identification is difficult if the genomic region is large. Ortutay and Vihinen [91] developed an *in silico* method for the identification of disease genes by integrating gene ontology and protein interaction networks. Their method identifies 26 novel primary immunodeficiency-related genes. To understand the disease phenotype in human diseases, Lee *et al.* [92] constructed the bipartite human disease association network in which nodes represent diseases and two diseases are linked if mutant enzymes associated with them catalyze the adjacent metabolic reaction. Their finding suggests that the connected disease pair shows high correlation in reaction flux rate. Moreover, the more connected a disease is to another disease, the higher its prevalence and associated mortality.

An interaction study of the drug-protein bipartite graph suggests the importance of anticancer drugs such as imatinib (marketed as Gleevec) and sunitinib (marketed as Sutent) [93] for the cancer treatment. This study highlights the properties of drug targets in the context of cellular networks and how it can be used for the understanding of a disease-gene product. The immune response can be seen as a dynamic interplay among interdependent interaction networks such as TF-gene interaction networks, cell-cell interactions, and cytokine interaction networks. To have a holistic understanding of immune response, one has to incorporate a local and systemic immune response to pathogens. The dynamical constraint-based modeling of such an immune network suggests the hijacking of a host immune molecular system for the persistence of disease in a bacterial infection [94].

Our ability to combat epidemic and gene-dependent diseases depends on a deep understanding of social and cellular network topologies. Moreover, the hubs of these cellular networks play a crucial role in disease development in animal models, implying a relationship between disease and the underlying biological processes. Such systematic studies can be extended to other diseases and organisms, for identifying important components in disease pathology.

## 37.4 FUTURE CHALLENGES AND SCOPE

The experimental data on which the modeling and analysis of biological networks are based are far from complete and contain high rates of false-positive errors. There is also a need for more accurate and reliable experimental methodology development for future research in cellular networks. Batada *et al.* [95] conclude that for the small interaction datasets, party hubs (which interact with all the partners simultaneously) cannot be distinguished from date hubs (which vary their connecting partners depending on time and location). The current view of the transcription regulatory network as an interplay between transcription factors and their corresponding genes is a highly simplified one. Actually, gene regulation can occur at many different stages such as the activation of the gene, synthesis rate, splicing, and efficacy of the export of RNA from the nucleus for its translation and final degradation in the cytoplasm. New scientific evidence suggests that information not only flows from DNA to RNA, but also can flow back from RNA to DNA to guide the developmental processes in an organism [96–98]. Many different elements such as miRNAs, temporal regulation, and transport proteins need to be integrated with the current gene regulatory network to provide a biologically meaningful description.

## ACKNOWLEDGMENTS

Gaurav Kumar acknowledges support from the *Macquarie University Research Scholarship* (MQRES) Award. This work also was supported by the award of the ARC Centre of Excellence in Bioinformatics grant (CE0348221) to S.R.

## REFERENCES

1. A.L. Barabasi. Linked: The New Science of Networks. Perseus Publishing, Cambridge, MA, 2002, p. 280.
2. P. Erdos and A. Renyi. On random graphs. *I. Publ. Math. Debrecen*, 6:290–297, 1959.
3. S. Miligram. The small world problem. *Psychol Today*, 2:60–67, 1967.
4. Small World Project. http://smallworld.columbia.edu/project.html.
5. E. Chacko and S. Ranganathan. *Graph Theory in Bioinformatics. Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications*. Wiley, New York, 193–219, 2011.
6. R. Albert, *et al.* Diameter of the World-Wide Web. *Nature*, 401:130, 1999.
7. F. Liljeros, *et al.* The web of human sexual contacts. *Nature*, 411(6840):907–908, 2001.
8. A.L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
9. A.L. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: The topology of the World-Wide Web. *Physica A*, 281:69–77, 2000.
10. A.L. Barabasi and E. Bonabeau. Scale-free networks. *Sci Am*, 288(5):60–69, 2003.
11. A.L. Barabasi and Z.N. Oltvai. Network biology: Understanding the cell's functional organization. *Nat Rev Genet*, 5(2):101–113, 2004.
12. H. Jeong, *et al.* Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.
13. H. Jeong, *et al.* The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000.
14. A. Wagner and D.A. Fell. The small world inside large metabolic networks. *Proc Biol Sci*, 268(1478):1803–1810, 2001.
15. S. Wuchty. Scale-free behavior in protein domain networks. *Mol Biol Evol*, 18(9):1694–1702, 2001.
16. B. Schwikowski, P. Uetz, and S. Fields. A network of protein-protein interactions in yeast. *Nat Biotechnol*, 18(12):1257–1261, 2000.
17. E.H. Davidson, *et al.* A genomic regulatory network for development. *Science*, 295(5560):1669–1678, 2002.
18. N.M. Luscombe, *et al.* Genomic analysis of regulatory network dynamics reveals large topological changes. *Nature*, 431(7006):308–312, 2004.
19. L.A. Amaral, *et al.* Classes of small-world networks. *Proc Natl Acad Sci U S A*, 97(21):11149–11152, 2000.
20. P. Erdos and A. Renyi. On the evolution of random graphs. *Publ Math Inst Hung Acad Sci*, 5:17–61, 1960.
21. D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.
22. D.J. Watts. Small worlds. The dynamics of networks between order and randomness. In *Princeton Studies in Complexity*. Princeton University Press, Princeton, NJ, 1999.
23. D.J. Watts. Networks, Dynamics, and the Small-World Phenomenon. *Am J Sociol*, 105(2):493–527, 1999.
24. R. Sedgewick. *Algorithms in C++*, 3rd edition, Graph Algorithms, Volume 2, part 5, Addison-Wesley, Reading, MA, 1998, pp. 1–495.

25. E. Ravasz, *et al.* Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555, 2002.
26. S. Maslov and K. Sneppen. Specificity and stability in topology of protein networks. *Science*, 296(5569):910–913, 2002.
27. A.H. Tong, *et al.* Global mapping of the yeast genetic interaction network. *Science*, 303(5659):808–813, 2004.
28. B. Vogelstein, D. Lane, and A.J. Levine. Surfing the p53 network. *Nature*, 408(6810):307–310, 2000.
29. O. Mason and M. Verwoerd. Graph theory and networks in biology. *IET Syst Biol*, 1(2):89–119, 2007.
30. L.C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.
31. R. Milo, *et al.* Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.
32. S.S. Shen-Orr, *et al.* Network motifs in the transcriptional regulation network of Escherichia coli. *Nat Genet*, 31(1):64–68, 2002.
33. S. Mangan, A. Zaslaver, and U. Alon. The coherent feedforward loop serves as a sign-sensitive delay element in transcription networks. *J Mol Biol*, 334(2):197–204, 2003.
34. S. Mangan and U. Alon. Structure and function of the feed-forward loop network motif. *Proc Natl Acad Sci U S A*, 100(21):11980–11985, 2003.
35. M. Khammash. Reverse engineering: the architecture of biological networks. *Biotechniques*, 44(3):323–329, 2008.
36. R. Milo, *et al.* Superfamilies of evolved and designed networks. *Science*, 303(5663): 1538–1542, 2004.
37. S. Wernicke. Efficient detection of network motifs. *IEEE/ACM Trans Comput Biol Bioinform*, 3(4):347–359, 2006.
38. S. Wernicke and F. Rasche. FANMOD: A tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153, 2006.
39. M.E.J. Newman. Detecting community structure in networks. *Eur Phys J B*, 38:321–330, 2004.
40. D.J. Watts, P.S. Dodds, and M.E. Newman. Identity and search in social networks. *Science*, 296(5571):1302–1305, 2002.
41. G. Palla, *et al.* Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.
42. M. Girvan and M.E. Newman. Community structure in social and biological networks. *Proc Natl Acad Sci U S A*, 99(12):7821–7826, 2002.
43. M.E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 69(2 Pt 2):026113, 2004.
44. A. Clauset, M.E. Newman, and C. Moore. Finding community structure in very large networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 70(6 Pt 2):066111, 2004.
45. A. Clauset, C. Moore, and M.E. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.
46. D. Segre, *et al.* Modular epistasis in yeast metabolism. *Nat Genet*, 37(1):77–83, 2005.
47. P. Uetz, *et al.* A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae. *Nature*, 403(6770):623–627, 2000.

48. T. Ito, *et al.* A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proc Natl Acad Sci U S A*, 98(8):4569–4574, 2001.
49. L. Giot, *et al.* A protein interaction map of Drosophila melanogaster. *Science*, 302(5651):1727–1736, 2003.
50. S. Li, *et al.* A map of the interactome network of the metazoan C. elegans. *Science*, 303(5657):540–543, 2004.
51. J.F. Rual, *et al.* Towards a proteome-scale map of the human protein-protein interaction network. *Nature*, 437(7062):1173–1178, 2005.
52. H. Hishigaki, *et al.* Assessment of prediction accuracy of protein function from protein–protein interaction data. *Yeast*, 18(6):523–531, 2001.
53. H.N. Chua, W.K. Sung, and L. Wong. Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. *Bioinformatics*, 22(13):1623–1630, 2006.
54. A. Vazquez, *et al.* The topological relationship between the large-scale attributes and local interaction patterns of complex networks. *Proc Natl Acad Sci U S A*, 101(52):17940–17945, 2004.
55. E. Nabieva, *et al.* Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 21 (Suppl 1):i302–i310, 2005.
56. B.P. Kelley, *et al.* Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc Natl Acad Sci U S A,* 100(20):11394–11399, 2003.
57. R. Sharan, *et al.* Conserved patterns of protein interaction in multiple species. *Proc Natl Acad Sci U S A*, 102(6):1974–1979, 2005.
58. H.J. Morowitz. A theory of biochemical organization, metabolic pathways and evolution. *Complexity*, 4:39–53, 1999.
59. E. Borenstein, *et al.* Large-scale reconstruction and phylogenetic analysis of metabolic environments. *Proc Natl Acad Sci U S A*, 105(38):14482–14487, 2008.
60. A. Kreimer, *et al.* The evolution of modularity in bacterial metabolic networks. *Proc Natl Acad Sci U S A*, 105(19):6976–6981, 2008.
61. C. Pal, B. Papp, and M.J. Lercher. Adaptive evolution of bacterial metabolic networks by horizontal gene transfer. *Nat Genet*, 37(12):1372–1375, 2005.
62. D. Segre, D. Vitkup, and G.M. Church. Analysis of optimality in natural and perturbed metabolic networks. *Proc Natl Acad Sci U S A*, 99(23):15112–15117, 2002.
63. I. Famili, *et al.* Saccharomyces cerevisiae phenotypes can be predicted by using constraint-based analysis of a genome-scale reconstructed metabolic network. *Proc Natl Acad Sci U S A*, 100(23):13134–13139, 2003.
64. J. Forster, *et al.* Large-scale evaluation of in silico gene deletions in Saccharomyces cerevisiae. *OMICS*, 7(2):193–202, 2003.
65. A.P. Burgard, P. Pharkya, and C.D. Maranas. Optknock: A bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnol Bioeng*, 84(6):647–657, 2003.
66. B. Papp, C. Pal, and L.D. Hurst. Metabolic network analysis of the causes and evolution of enzyme dispensability in yeast. *Nature*, 429(6992):661–664, 2004.
67. D. Vitkup, P. Kharchenko, and A. Wagner. Influence of metabolic network structure and function on enzyme evolution. *Genome Biol*, 7(5):R39, 2006.

68. Y. Makita, *et al.* DBTBS: Database of transcriptional regulation in Bacillus subtilis and its contribution to comparative genomics. *Nucleic Acids Res*, 32(Database issue):D75–D77, 2004.
69. V. Matys, *et al.* TRANSFAC and its module TRANSCompel: Transcriptional gene regulation in eukaryotes. *Nucleic Acids Res*, 34(Database issue):D108–D110, 2006.
70. R.V. Davuluri, *et al.* AGRIS: Arabidopsis gene regulatory information server, an information resource of Arabidopsis cis-regulatory elements and transcription factors. *BMC Bioinformatics*, 4:25, 2003.
71. V.V. Svetlov and T.G. Cooper. Review: compilation and characteristics of dedicated transcription factors in *Saccharomyces cerevisiae*. *Yeast*, 11(15):1439–1484, 1995.
72. C. Rodriguez-Caso, M.A. Medina, and R.V. Sole. Topology, tinkering and evolution of the human transcription factor network. *FEBS J*, 272(24):6423–6434, 2005.
73. C.L. Wei, *et al.* A global map of p53 transcription-factor binding sites in the human genome. *Cell*, 124(1):207–219, 2006.
74. C.E. Horak and M. Snyder. ChIP-chip: A genomic approach for identifying transcription factor binding sites. *Metho Enzymol*, 350:469–483, 2002.
75. Z. Bar-Joseph, *et al.* Computational discovery of gene modules and regulatory networks. *Nat Biotechnol*, 21(11):1337–1342, 2003.
76. T.I. Lee, *et al.* Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science*, 298(5594):799–804, 2002.
77. S. Balaji, *et al.* Comprehensive analysis of combinatorial regulation using the transcriptional regulatory network of yeast. *J Mol Biol*, 360(1):213–227, 2006.
78. G. Manning, *et al.* Evolution of protein kinase signaling from yeast to man. *Trends Biochem Sci*, 27(10):514–520, 2002.
79. G. Weng, U.S. Bhalla, and R. Iyengar. Complexity in biological signaling systems. *Science*, 284(5411):92–96, 1999.
80. S. Li, S.M. Assmann, and R. Albert. Predicting essential components of signal transduction networks: A dynamic model of guard cell abscisic acid signaling. *PLoS Biol*, 4(10):e312, 2006.
81. E. Eisenberg and E.Y. Levanon. Preferential attachment in the protein network evolution. *Phys Rev Lett*, 91(13):138701, 2003.
82. G. Bianconi and A.L. Barabasi. Competition and multiscaling in evolving networks. *Europhys Lett*, 54(4):436–442, 2001.
83. H.B. Fraser, *et al.* Evolutionary rate in the protein interaction network. *Science*, 296(5568):750–752, 2002.
84. A. Vazquez, *et al.* Modeling of protein interaction networks. *ComPlexUs*, 1:38–44, 2003.
85. M.N. McClean, *et al.* Cross-talk and decision making in MAP kinase pathways. *Nat Genet*, 39(3):409–414, 2007.
86. R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Phys Rev Lett*, 86(14):3200–3203, 2001.
87. R.M. May and A.L. Lloyd. Infection dynamics on scale-free networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 64(6 Pt 2):066112, 2001.
88. H. Hethcote. The mathematics of infectious disease. *SIAM Rev*, 42(4):599–653, 2000.
89. R. Pastor-Satorras and A. Vespignani. Immunization of complex networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 65(3 Pt 2A):036104, 2002.

90. G. Bohn, *et al.* A novel human primary immunodeficiency syndrome caused by deficiency of the endosomal adaptor protein p14. *Nat Med*, 13(1):38–45, 2007.

91. C. Ortutay and M. Vihinen. Identification of candidate disease genes by integrating gene ontologies and protein-interaction networks: Case study of primary immunodeficiencies. *Nucleic Acids Res*, 37(2):622–628, 2009.

92. D.S. Lee, *et al.* The implications of human metabolic network topology for disease comorbidity. *Proc Natl Acad Sci U S A*, 105(29):9880–9885, 2008.

93. M.A. Yildirim, *et al.* Drug-target network. *Nat Biotechnol*, 25(10):1119–1126, 2007.

94. J. Thakar, *et al.* Constraint-based network model of pathogen-immune system interactions. *J R Soc Interface*, 6(36):599–612, 2009.

95. N.N. Batada, *et al.* Stratus not altocumulus: A new view of the yeast protein interaction network. *PLoS Biol*, 4(10):e317, 2006.

96. R.C. Lee and V. Ambros. An extensive class of small RNAs in Caenorhabditis elegans. *Science*, 294(5543):862–864, 2001.

97. J.S. Mattick. The genetic signatures of noncoding RNAs. *PLoS Genet*, 5(4):e1000459, 2009.

98. T. Reguly, *et al.* Comprehensive curation and analysis of global interaction networks in Saccharomyces cerevisiae. *J Biol*, 5(4):11, 2006.

99. X. Zhu, M. Gerstein, and M. Snyder. Getting connected: Analysis and principles of biological networks. *Genes Dev*, 21(9):1010–1024, 2007.

# 38

# PROBABILISTIC APPROACHES FOR INVESTIGATING BIOLOGICAL NETWORKS

Jérémie Bourdon and Damien Eveillard

Last decade saw a significant increase of high throughput experiments. As a major achievement, these novel techniques replicate the molecular experiments, which opens perspectives of quantitative behavior investigations. For illustration, it is now possible to define the concentration for which a protein (*i.e.*, transcription factor) may activate a given gene. This information used to be considered as a limitative factor for producing accurate dynamical models of large biological regulatory networks [5]. Today, one must take it into account for building large quantitative models. Furthermore, high throughput experiments describe, as well, macromolecular processes via their temporal properties. Thus, biological processes can be summarized by the evolutions of their biological compounds over time (*i.e.,* a succession of biological qualitative states or temporal patterns). Such experiments show temporal parameters that refine, in a natural manner, the qualitative models describing biological systems. However, these refinements, which present great biological interests, raise similarly several computational concerns. One is dealing with the complexity that originates from the large amount of experimental data. The challenge hence consists in trimming the experimental information at disposal for extracting the major driving compounds and their respective interactions within a network. Another is taking into account the quantitative impacts of these components on the biological

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

dynamics (in [3], the authors propose a probabilistic model allowing us to introduce a quantification under the hypothesis that the global behavior of a quantity is an accumulation of small variations). It implies dealing with different ranges of concentration variations for several compounds. Such integrations rise the opportunity to build predicting models that reproduce replicated experimental results. A third issue is integrating temporal behaviors in such complex systems (a study of the temporal effects in *Escherichia coli* carbon starvation system can be found in [1]). In this case, the complexity lies in introducing partial informations. Indeed, a temporal behavior of a given compound, like a gene, is often known, whereas others remains not well understood.

To sum up, current computational challenges are (i) analyzing large amount of data and their inherent complexity, introducing both (ii) quantitative knowledge and (iii) temporal properties into models of biological regulatory networks. Among the computational biology techniques, probabilistic approaches appear as an accurate consensus that deal with those features. This chapter proposes a short overview of their applications for investigating biological regulatory networks. We will first present the theoretical framework needed for such particular biological systems (Section 38.1). It emphasizes a qualitative modeling that comes from both empirical and experimental knowledge. Second, we will show (Section 38.2) an overview of the analyses that can be performed on such a kind of model.

## 38.1 PROBABILISTIC MODELS FOR BIOLOGICAL NETWORKS

Several probabilistic approaches exist, and they are not all accurate for modeling dynamical biological systems. Feedback loops are the most common control process of natural systems, especially for gene regulatory networks (see Figure 38.1). Taking such loops into account (*i.e.*, positive or negative) is therefore the major criterion for choosing one probabilistic approach among others.

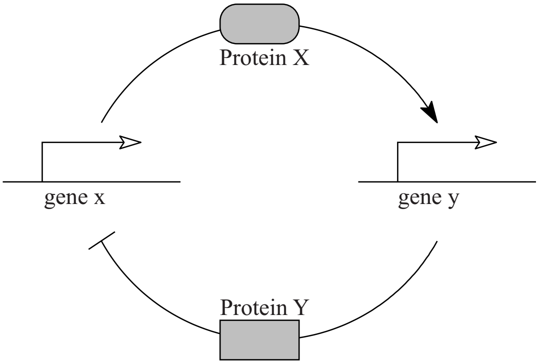


**Figure 38.1** Representation of a gene regulatory network with two genes. Gene x activates the transcription of gene y via the production of protein X. Reversely, gene y represses the transcription of gene x via the production of protein Y.

Because it is based on Boolean networks, probabilistic Boolean networks (PBNs) [19] is a probabilistic framework that deals with feedback loops. This modeling gives two great advantages. First, it allows a qualitative analysis [4, 5] by investigating the qualitative properties of the "Boolean core." Second, it proposes a quantitative analysis when focusing on the probabilities. Combining probabilities and BN logical informations gives thus insights about both temporal behaviors and predictions of biological compounds quantities, which comes up to recent biological expectations. In this section, we focus on Boolean networks.

### 38.1.1 Boolean Networks

The Boolean networks represent the qualitative core of the PBN. Their interpretation constitutes a key step for a complete understanding of their probabilistic extensions. Introduced by Stuart Kauffman and co-workers [8, 13], the Boolean networks quickly raise a strong interest from both physical and biological fields. Their applications in computational biology resume the genes by switches: The gene activity can be either ON or OFF. This assumption comes from a simplification of the step function that represents the activation of a gene by another. Because these genes interact on each other, their interactions build a network, in which the evolution of a given gene activity depends on the activities of other genes (see [5] for review). Following this assumption, a Boolean network can be seen as a vector of Boolean functions, such as follows.

**Definitio 38.1** *A Boolean network $B = (V, F)$ is a pair, where*

- *$V = \{x_1, \ldots, x_n\}$ is a set of Boolean variables (i.e., genes), $\forall i, x_i \in \{0, 1\}$.*
- *$F = \{f_1, \ldots, f_n\}$ is a set of Boolean functions, $\forall i, f_i : \{0, 1\}^n \rightarrow \{0, 1\}$. Here, $f_i$ describes the evolution of gene $i$.*

Such networks allow the dynamical description of given phenomena. Formally, if $X(t) = (x_1(t), \ldots, x_n(t))$ represents the value of all variables at time $t$, then $X(t + 1) = (x_1(t + 1), \ldots, x_n(t + 1))$, where $\forall i, x_i(t + 1) = f_i(x_1(t), \ldots, x_n(t))$ is the value of all variables after one iteration of the Boolean network. Note that for $n$ genes, the corresponding number of Boolean networks is $(2^n)^{(2^n)}$. Among them, few are accurate with biological knowledge. Their identifications is therefore a major issue of Boolean networks theory.

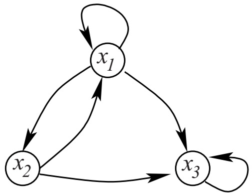
*Dependency Graph.* We are interested in trimming the number of Boolean networks of interest. In general cases, $f_i$ depends on a subset of the Boolean variables only. A Boolean variable $x_j$ is so-called *ficticius* for $f_i$ if, and only if, for all $(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) \in \{0, 1\}^{n-1}$:

$$f_i(x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n) = f_i(x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_n)$$

It shows that some genes are not useful for predicting the behaviors of the system: $f_i$ does not depend on variable $x_j$, or in other words, $\partial f_i(x_1, \ldots, x_n)/\partial x_j = 0$. All other genes are so-called *essential* because they impact the dynamical description. Thus, knowing the values of $f_i$ for all possible affectations of essential variables is sufficient. Furthermore, it is informative to draw the Boolean variables dependencies using a directed graph.

**Definitio 38.2** *Let $B = (V, F)$ a Boolean network. The dependency graph $G = (V, E)$ of $B$ is defined by $(j, i) \in E$ if, and only if, $x_j$ is an essential variable for $f_i$.*

The following figure illustrates an example of dependency graph. Herein, $x_1$ and $x_2$ are essential for $f_1$; $x_1$ is essential for $f_2$; and $x_1$, $x_2$, and $x_3$ are essential for $f_3$.



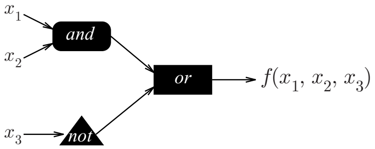
*Representations of Boolean Functions.* The Boolean networks can be complex, and one of the major concerns remains the storage of the Boolean functions. Indeed, it is necessary to store one or more Boolean function per gene. Several ways define efficiently and unambiguously a Boolean function.

TRUTH TABLE. The simplest way to define a Boolean function consists in providing its truth table.

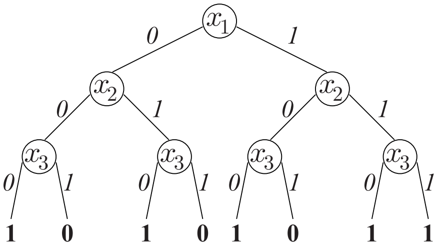
| $x_1$ | $x_2$ | $x_3$ | $f(x_1, x_2, x_3)$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

However, based on the function values for all its possible entries, the size of such a table is exponential.

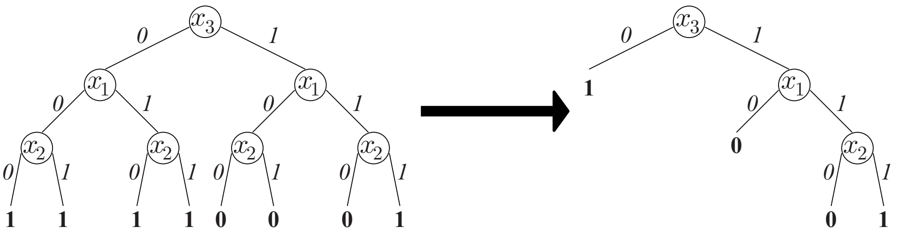
LOGIC EXPRESSIONS. Any Boolean function can be expressed via simple Boolean operations $\{\wedge$(logical and), $\vee$(logical or)$and\neg$(logical not)$\}$. Combining these logical operators is equivalent to design a logic circuit that mimics a given Boolean function. It compacts the expression of the Boolean function. Like this, the previous truth table corresponds to $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee \neg x_3$, or this logical circuit:



BINARY DECISION TREES. Another expression of the truth table relies on a branching process. As illustration, based on the previous truth table, the first half of the table corresponds to all entries when the first variable is false. By using this consideration recursively, one defines a binary decision tree that matches the truth table.



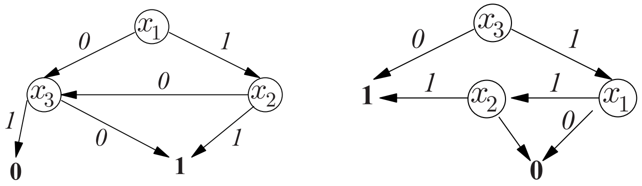
Although the size of such a graph is still exponential, it allows us to think about some improvements. Like this, it emphasizes that some part of the decision tree are not that informative, which it is not obvious when reading the truth table only. Indeed, if $x_1 = x_2 = 1$ is true, then for any choice of $x_3$, $f(x_1, x_2, x_3) = 1$. As a consequence, the last part of the tree can be pruned and replaced by a leaf that has value 1. The variable ordering is therefore one of the key features. For illustration, the decision tree can be simplified by considering the ordering $(x_3, x_1, x_2)$ instead of $(x_1, x_2, x_3)$. In this case, half of the binary decision tree is pruned.



This tree manipulation represents a natural way to emphasize an information of interest. Lee [14] extends this approach and proposes a binary decision diagram (BDD). It is a directed acyclic graph obtained when applying two simplification rules to the decision tree:

1. Merging any isomorphic subgraphs.
2. Eliminating any node whose two childs are isomorphic.

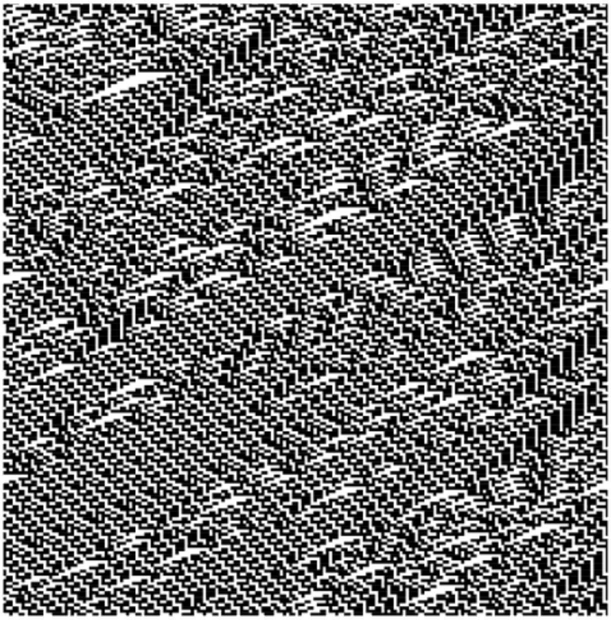
When applied on the above Decision tree (respectively, for the orders $(x_1, x_2, x_3)$ and $(x_3, x_1, x_2)$), these two rules give the following binary decision diagrams.



These diagrams show different roots; respectively $x_1$ for the order $(x_1, x_2, x_3)$ and $x_3$ for $(x_3, x_1, x_2)$, but both show similar decision patterns and truth tables, as shown above. As a concrete application in the computational biology field, A. Naldi and co-workers [17] propose a similar approach that successfully investigates gene regulatory network Boolean models.

*Examples of Boolean Networks.* A literature review shows several modelings of biological systems using Boolean networks and, moreover, plenty modeling approaches that derive from Boolean networks. We propose to present herein three of them.

CELLULAR AUTOMATA. As a classical Boolean network extension, a cellular automaton [10] is a modeling approach that focuses on the notion of variable locality. Indeed, the cellular automata consider a unique Boolean function $f(z_1, \dots, z_k)$ in $k$ variables and a set of $k$ integers $\mathcal{N} = \{n_1, \dots, n_k\} \subset \mathbb{Z}$ that corresponds to the neighborhood to be considered. The associated Boolean network is then defined by setting $f_i(x_1, \dots, x_n) = f(x_{i+n_1}, \dots, x_{i+n_k})$. Notice here that the indices belong to the torus $\{1, \dots, n\}$ (*i.e.,* all operations are assumed to be modulo $n$), where $x[n]$ denotes the remainder of the ordinary euclidean division of $x$ by $n$. The following figure shows a trajectory of the cellular automaton corresponding to $\mathcal{N} = \{-1, 0, 1\}$ and $f(x_1, x_2, x_3) = (\neg x_3 \wedge x_2) \vee (x_3 \wedge \neg x_1) \vee (x_3 \wedge x_1 \wedge \neg x_2)$. More precisely, the first line represents the initial values of all variables (pixels in white if 0 or in black if 1). Each line below is computed from the previous one and the Boolean network.



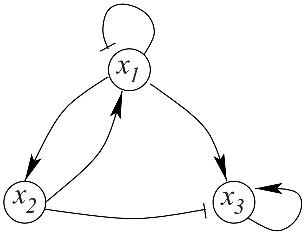
As a major feature, the cellular automaton theory achieves the identification of periodic patterns that can be derived from the evolution of the automaton. For illustration, the above picture shows periodic patterns, like the tiny white triangles that are repeated on the diagonals of the figure. They characterize one of the specific properties of the Boolean functions. They might be automatically extracted from the simulation traces via standard pattern finding approaches.

DISCRETE NETWORKS. The major biological assumption depicted within the Boolean networks remains the discrete abstraction of the gene activity. However, in many cases, assuming a gene as a simple interruptor is not appropriate. It has been noticed that genes may have different behaviors depending on their activation levels [6]. Other genes may present different promotors that respond differentially in function of the transcription factor activity level [18]. Therefore, a single Boolean domain for each variable is no longer sufficient. A natural way to deal with this consists in introducing thresholds for modeling gene activity, in order to differentiate several effects of the gene activity. Following this assumption, R. Thomas and co-workers [26, 24] propose a model of discrete networks where the variables are discretized. D. Thieffry and co-workers extend this idea [4, 25, 23]. It allows them to analyze the qualitative properties of the discrete models, which bridges the gap between continuous models (*i.e.,* ordinary differential equation based models) and Boolean networks. The implementation of such an approach shows fine qualitative results [16] but reversely produces discrete exponential graphs. Hence, they are often too large for allowing a probabilistic extension on the discrete networks.

INFLUENCE GRAPHS. Following similar gene regulatory network modeling motivations, A. Siegel and co-workers [22] propose an another extension with a restricted type of Boolean functions. Indeed, they consider a function $f_i(x_1, \ldots, x_n) = y_1 \vee \cdots \vee y_n$, where $y_k = 0$ if variable $x_k$ is ficticious for $f_i$; and $y_k = x_k$ or $y_k = \neg x_k$, where $x_k$ is essential for $f_i$. The choice between these two cases relies on the fact that $x_k$ is an activator or a repressor for $x_i$. As a direct consequence, each essential variable appears only once in the Boolean function expression, either as an activator or as an inhibitor. From this compact expression of the gene regulatory network, they define a so-called *generalized dependency graph* $\widetilde{G} = (V, E)$ of the Boolean network $B = (V, F)$, where there exists an edge $(i, j, s) \in E \subset V \times V \times \{-, +\}$ if, and only if, $x_j$ is an essential variable for $f_i$ and $s = \text{sign}\,(\partial f_i(x_1, \ldots, x_n)/\partial x_j)$. For illustration, when considering three Boolean functions that represent a given biological knowledge:

$$\begin{aligned} f_1(x_1, x_2, x_3) &= \neg x_1 \vee x_2 \\ f_2(x_1, x_2, x_3) &= x_1 \\ f_3(x_1, x_2, x_3) &= x_1 \vee \neg x_2 \vee x_3 \end{aligned}$$

We can picture these constraints by the following generalized dependency graph:



This graph represents an elegant summary of the logical (*i.e.*, qualitative) properties that emerge from the set of biological constraints. Note that these Boolean constraints might be automatically investigated. In particular, P. Veber and co-workers [27] propose an *in silico* protocol that checks the consistencies of these constraints with experimental knowledges at disposal. It represents an elegant validation of the qualitative properties of the "*Boolean core*" before further probabilistic extensions.

### 38.1.2 Probabilistic Boolean Networks: A Natural Extension

Boolean networks do not always reflect the correct behaviors of complex biological models. In fact, at some point, quantitative models need flexibility for taking into account the inherent complexity of gene interactions (*i.e.*, nonlinearity due to post-transcriptional regulations) or to deal with incomplete data. In this purpose, Shmulevich and co-workers [19] introduced PBNs, that is, a probabilistic extension of Boolean networks.

**Definitio 38.3** *A probabilistic Boolean network $B = (V, \mathcal{F})$ is a pair, where*

- *$V = \{x_1, \ldots, x_n\}$ is a set of Boolean variables (i.e., genes), $\forall i, x_i \in \{0, 1\}$.*
- *$\mathcal{F} = \{F_1, \ldots, F_n\}$ is a set, where $F_i = \{(f_i^{(1)}, p_i^{(1)}), \ldots, (f_i^{(l_i)}, p_i^{(l_i)})\}$ is a set of pairs composed by a Boolean function and a probability. For all i, one has $\sum_{k\in\{1,\ldots,l_i\}} p_i^{(1)} = 1$. Here, the evolution of gene i is predicted by $f_i^{(k)}$ with probability $p_i^{(k)}$.*

The dynamics of the biological system are now described using Boolean random variables $(X_1(t), \ldots, X_n(t))$, which satisfy:

$$\forall i, \forall k \in \{1, \ldots, l_i\}, \text{Prob}\{X_i(t+1) = f_i^{(k)}(X_1(t), \ldots, X_n(t))\} = p_i^{(k)}$$

From the simulation viewpoint, it opens other perspectives. If $(x_1, \ldots, x_n)$ is the current Boolean affectation of all gene activities, the activity of gene $i$ becomes $f_i^{(k)}(x_1, \ldots, x_n)$ with probability $p_i^{(k)}$. Like this, the probabilities add an uncertainty feature to the model. Intuitively, one disposes of several predictors for a gene activity. One can trust a predictor with a given probability.

Note that PBN can be decomposed as a finite set of $\prod_{i=1}^{n} |F_i|$ constituent Boolean networks with some transitions probabilities between them that are determined by the predictor probabilities.

### 38.1.3 Inferring Probabilistic Models from Experiments

The concern is to fit the previous probabilistic framework with biological knowledge at disposal. Several approaches have been proposed to build them from experiments using an automatic manner [19, 20, 21]. Mainly, experiments correspond to long time courses $C = ((x_1(1), \ldots, x_n(1)), \ldots, (x_1(T), \ldots, x_n(T)))$ involving measures of $n$ genes at $T$ different time steps. The goal is to build a PBN that reproduces $C$ as a trajectory with a high probability. Notice that if the time course originates from a single Boolean network, classical methods such as Viterbi or Baum-Welch algorithms can be adapted from the inference of hidden Markov model herein for contructing the most probable Boolean network. We observe that a PBN can be seen as a composition of a finite number of constituent Boolean networks. In this context, perturbation probabilities allow us to swap from a Boolean network to another. Based on this observation, the inference of a proper PBN from long time courses consists in three distinct steps:

1. First separate the time course into subsequences originating from the same constituent Boolean network.

2. Infer separately every constituent Boolean networks.
3. Retrieve the swapping probabilities between constituent Boolean networks, and construct the PBN.

Notice that the first step is crucial. The choice of the optimization method is also a major factor to ensure a convergence to the right PBN.

## 38.2 INTERPRETATION AND QUANTITATIVE ANALYSIS OF PROBABILISTIC MODELS

Previous analyses of the Boolean core allow us to investigate the qualitative properties of the biological regulatory networks. Others approaches exist for investigating both temporal and quantitative properties that emerge from models probabilistically extended.

### 38.2.1 Dynamical Analysis and Temporal Properties

One of the major biological expectations when studying regulatory networks is to extract general properties from the evolution of gene activities. This evolution is inherently encoded by the network. It can be represented by a dynamical graph (or state space) defined as follows:

**Definitio 38.4** *Let $B = (V, F)$ be a Boolean network and $n = |V|$. The dynamical graph $\mathcal{G} = (\{0, 1\}^n, E)$ associated with $B$ is a directed graph possessing an edge from $(x_1, \ldots, x_n)$ to $(x'_1, \ldots, x'_n)$ if this edge defines a possible update (several update strategies are described in the sequel).*

The dynamical graph may show attractors that represent key features of the system dynamic. It has been shown that phenotypes are very often associated with these attractors [11, 12]. Therefore, studying these properties of the dynamical graph presents great interest in a biological context. It explains why several approaches have been proposed. They focus on distinct viewpoints: simulation of the steady state distribution, algorithms from the graph theory to study the topological aspects of graphs, and model checking approaches. Since they introduce time, all these approaches are sensitive to the update of the biological system. For the same network, several update strategies were proposed. It results in distinct dynamical graphs.

***38.2.1.1 Synchronous Update.*** It is the most natural update. It corresponds to a succession of observations of the gene activities at some fixed time. The synchronous update strategy assumes that all genes are updated at the same time. Like this, if $(x_1(t), \ldots, x_n(t))$ is the Boolean affectation of all gene activities at time $t$, $(x_1(t+1), \ldots, x_n(t+1))$, where for all $i$, $x_i(t+1) = f_i(x_1(t), \ldots, x_n(t))$, is the Boolean affectation of all gene activities at time $t + 1$. For illustration, the

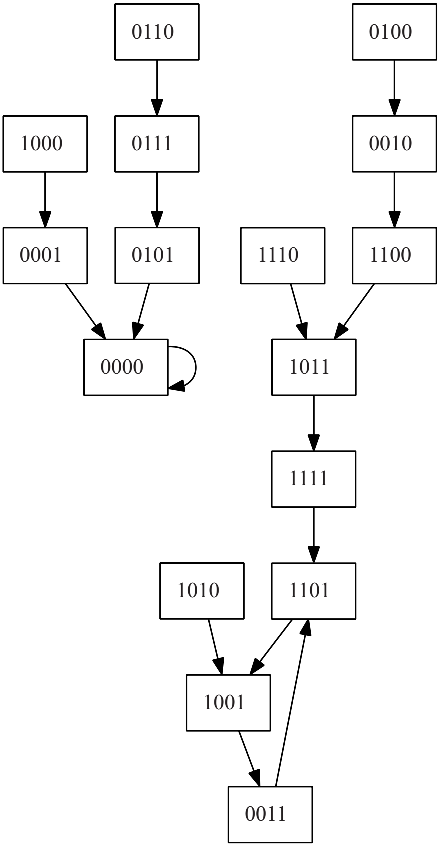


**Figure 38.2** Dynamical graph with a synchronous update strategy.

Figure 38.2 draws the synchronous dynamical graph associated with the Boolean network:

$$
\begin{aligned}
f_1(x_1, x_2, x_3, x_4) &= x_1 \wedge x_2 \vee \neg x_2 \wedge x_3 \\
f_2(x_1, x_2, x_3, x_4) &= x_1 \wedge x_3 \wedge x_4 \vee \neg x_1 \wedge x_2 \\
f_3(x_1, x_2, x_3, x_4) &= x_1 \wedge \neg x_2 \wedge x_4 \vee \neg x_2 \wedge \neg x_4 \\
f_4(x_1, x_2, x_3, x_4) &= x_1 \vee x_2 \wedge x_3 \vee \neg x_3 \wedge x_4
\end{aligned}
$$

***38.2.1.2 Asynchronous Update.*** One might observe that it is rare that two independent genes change their activity level together at the same time. Regarding the dynamical evolution of discretizations of continuous variables, this observation is inconsistent with the previous synchronous update strategy. We must suppose herein that only one gene can change at a given time. Like this, if $(x_1(t), \ldots, x_n(t))$ is the Boolean affectation of all gene activities at time $t$, there exists at most $n$ possible

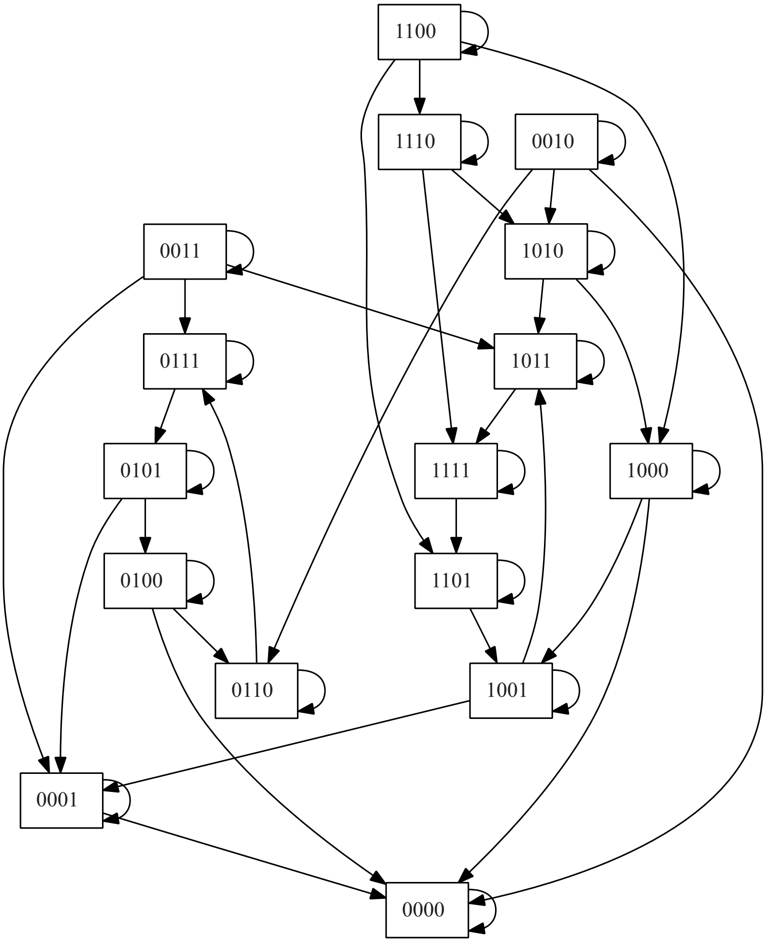


**Figure 38.3** Dynamical graph with an asynchronous update strategy.

affectations $(x_1(t+1), \ldots, x_n(t+1))$, where $x_i(t+1) = f_i(x_1(t), \ldots, x_n(t))$ if $i = j$ and $x_i(t+1) = x_i(t)$ otherwise, for all $j \in \{1, \ldots n\}$. For illustration, Figure 38.3 pictures the dynamical graph of the Boolean network above, with an asynchronous strategy.

***38.2.1.3 Mixed Updates with Priorities.*** Biological systems often need plasticity in their update strategies. Indeed, some genes are co-regulated, whereas others are independent. Thus, the asynchronous assumption is not fulfilled because of regulation specificities or an incomplete knowledge. In this context, Naldi and co-workers [16] proposed a mixed strategy that combines synchronous updates for genes having a similar regulation speed and asynchronous updates for others. For that, they define a synchronization partition of genes. Let $P = \{I_1, \ldots, I_m\}$, be a disjoint partition of $\{1, \ldots, n\}$ composed by nonempty sets (*i.e.,* $\forall u, I_u \neq \emptyset$,

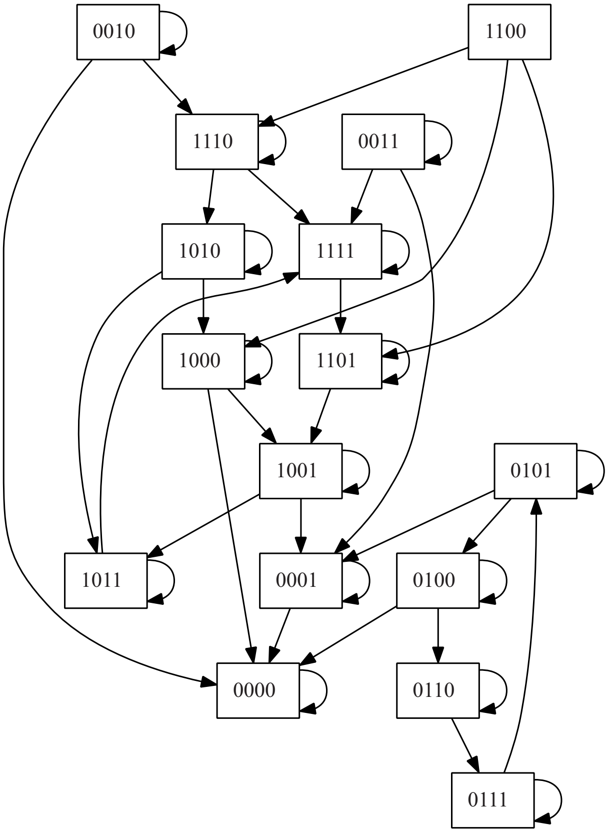


**Figure 38.4** Dynamical graph with a mixed update strategy with synchronization partition $P = \{\{1, 2\}, \{3\}, \{4\}\}$.

$\bigcup_{u=1}^{m} I_u = \{1, \ldots, n\}$ and $\forall u \neq v,\ I_u \cap I_v = \emptyset$). This partition describes synchronizations between genes. The two extremal cases correspond to $P = \{\{1, \ldots, n\}\}$ for synchronous updates and $P = \{\{1\}, \ldots, \{n\}\}$ for asynchronous updates. Then, if $(x_1(t), \ldots, x_n(t))$ is the Boolean affectation of all gene activities at time $t$, there exists at most $m$ possible affectations $(x_1(t+1), \ldots, x_n(t+1))$, where for all $u \in \{1, \ldots, m\}$, $x_i(t+1) = f_i(x_1(t), \ldots, x_n(t))$ if $i \in I_u$ and $x_i(t+1) = x_i(t)$ otherwise. For illustration, Figure 38.4 represents the dynamical graph of the previous Boolean network with a mixed update strategy with synchronization partition $P = \{\{1, 2\}, \{3\}, \{4\}\}$.

### 38.2.2 Impact of Update Strategies for Analyzing Probabilistic Boolean Networks

In a probabilistic context, the edges of a dynamical graph are endowed with a transition probability. Consequently, the dynamical graph becomes a Markov chain. Naturally, update strategies that impact the Boolean network interpretation play a major role for investigating probabilistic Boolean networks too. We propose to illustrate this point by showing how mixed updates are extended in the probabilistic context of PBN. Here, the dynamical graphs become Markov chains with $2^n$ states, whose transition matrix $T = (p_{i\to j})_{i,j\in(\{0,1\}^n)^2}$ is defined as follows.

Let $P = \{I_1, \dots, I_m\}$ be a synchronization partition of $\{1, \dots, n\}$. For all $u \in \{1, \dots, m\}$, let $I_u = \{s_1, \dots, s_{p_u}\}$ and define the sets

- $H_u = \prod_{s\in I_u}\{1, \dots, l_s\}$, where $l_s$ is the number of statistical predictors of gene $s$.
- $U_{i,j,u} = \{(k_{s_1}, \dots, k_{s_{p_u}}) \in H_u, (x_1, \dots, x_n) = i, x'_s = f_s^{(k_s)}(x_s)$ if $s \in I_u$ and $x'_s = x_s$ otherwise and $(x'_1, \dots, x'_n) = j\}$.

Then, if one has

$$p_{i\to j} = \sum_{u=1}^{m} \sum_{\substack{(k_{s_1}, \dots, k_{s_{p_u}}) \in U_{i,j,u}, \\ I_u = \{s_1, \dots, s_{p_u}\}}} \prod_{s\in I_u} p_s^{(k_s)}$$

This framework achieves a probabilized dynamical graph for all update strategies applied on a PBN. For illustration, let us consider PBN defined by

$$\begin{aligned}
f_1^{(1)}(x_1, x_2, x_3, x_4) &= x_1 \wedge x_2 \vee \neg x_2 \wedge x_3 & p_1^{(1)} &= 0.3\\
f_1^{(2)}(x_1, x_2, x_3, x_4) &= \neg x_1 \wedge x_2 \vee x_3 \wedge x_4 & p_1^{(2)} &= 0.7\\
f_2^{(1)}(x_1, x_2, x_3, x_4) &= x_1 \wedge x_3 \wedge x_4 \vee \neg x_1 \wedge x_2 & p_2^{(1)} &= 1\\
f_3^{(1)}(x_1, x_2, x_3, x_4) &= x_1 \wedge \neg x_2 \wedge x_4 \vee \neg x_2 \wedge \neg x_4 & p_3^{(1)} &= 1\\
f_4^{(1)}(x_1, x_2, x_3, x_4) &= x_1 \vee x_2 \wedge x_3 \vee \neg x_3 \wedge x_4 & p_4^{(1)} &= 1
\end{aligned}$$

This PBN is a generalization of the BN previously presented, the only change being on the rule for gene 1.

Figures 38.5, 38.6, and 38.7 represent the dynamical graphs with probabilities on edges for distinct strategy updates, respectively, synchronous, asynchronous, and mixed strategy with $P = \{\{1, 2\}, \{3\}, \{4\}\}$.

### 38.2.3 Simulations of a Probabilistic Boolean Network

Previous approaches emphasize the qualitative properties of the probabilistic Boolean networks. However, biologists might be interested by quantifying each biological compounds that interact within the biological network. Simulations represent a natural way to predict the biological compounds quantities. Note here that such a quantitative information is in accordance with the qualitative results previously shown. The quantitative information results from the transitions taken more or less

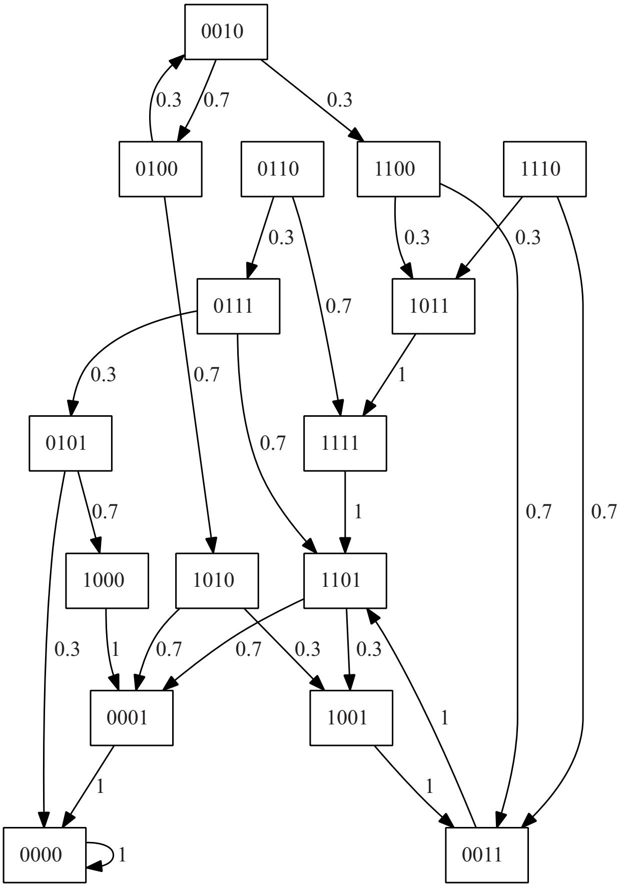


**Figure 38.5** Dynamical graph of a PBN with a synchronous strategy.

in a manner that is conformed with probabilities on the network transitions. In other words, qualitative properties indicate the potential qualitative transitions, whereas quantitative features represent the integration of all qualitative transitions. The Monte Carlo approach achieves such an integration by computing numerical values, given a probability distribution. In the PBN context, probabilities are related to the interactions. When one stays at one state in the graph, the Monte Carlo algorithm indicates what is the transition to take, in accordance with the probabilities associated with the transitions that go out from the given state (see Figure 38.8 for illustration). Following a Bernoulli law and a significant number of random walks through the graph, one can estimate the distribution of the biological quantities. In this context, we describe here a simulation based on a Markov chain Monte Carlo approach that estimates the equilibrium distribution. Several algorithms implement distinct approaches, among which Metropolis-Hasting algorithm and Gibbs sampling, are the most applied in computational biology (see [2] for details).

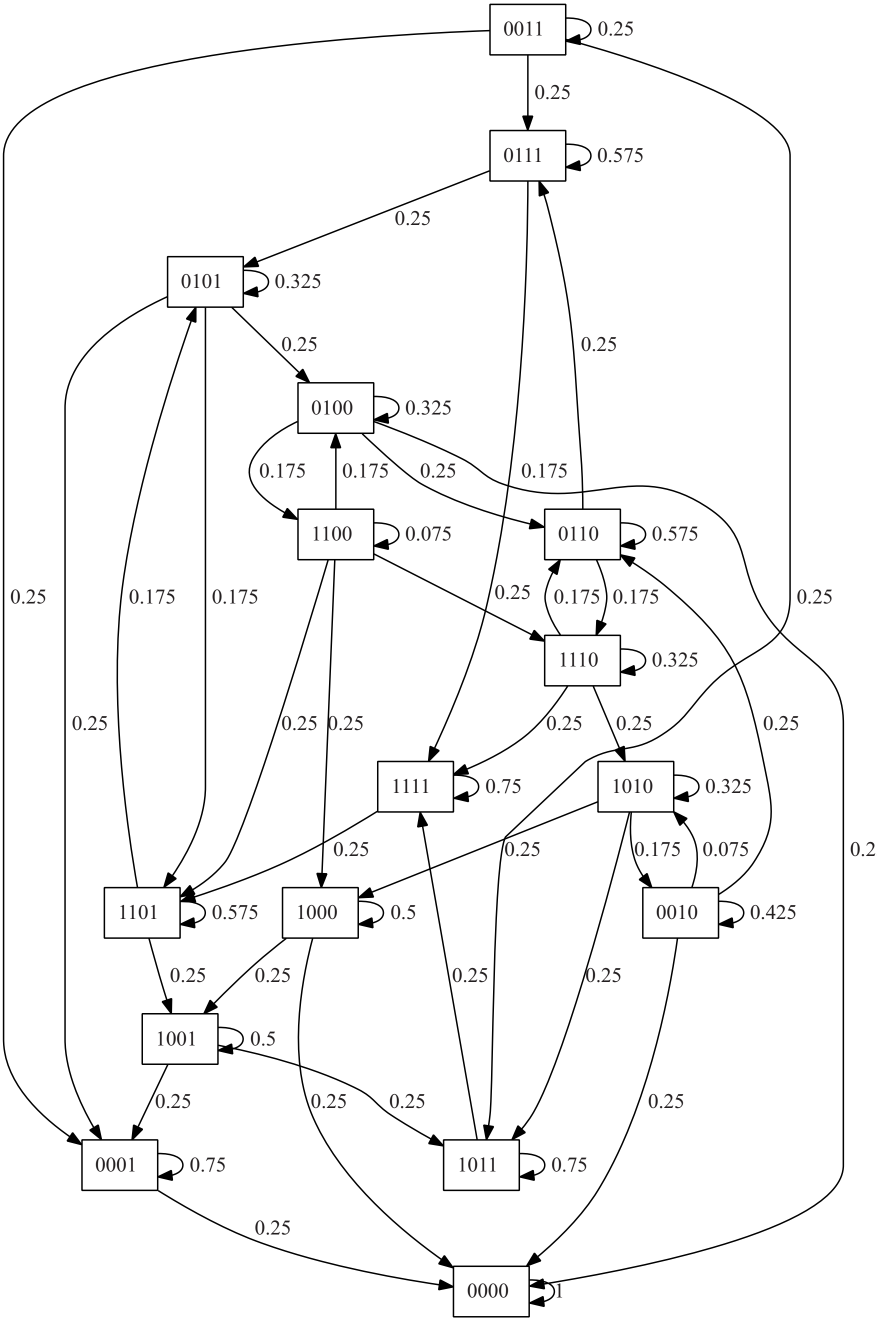


**Figure 38.6** Dynamical graph of a PBN with an asynchronous strategy.

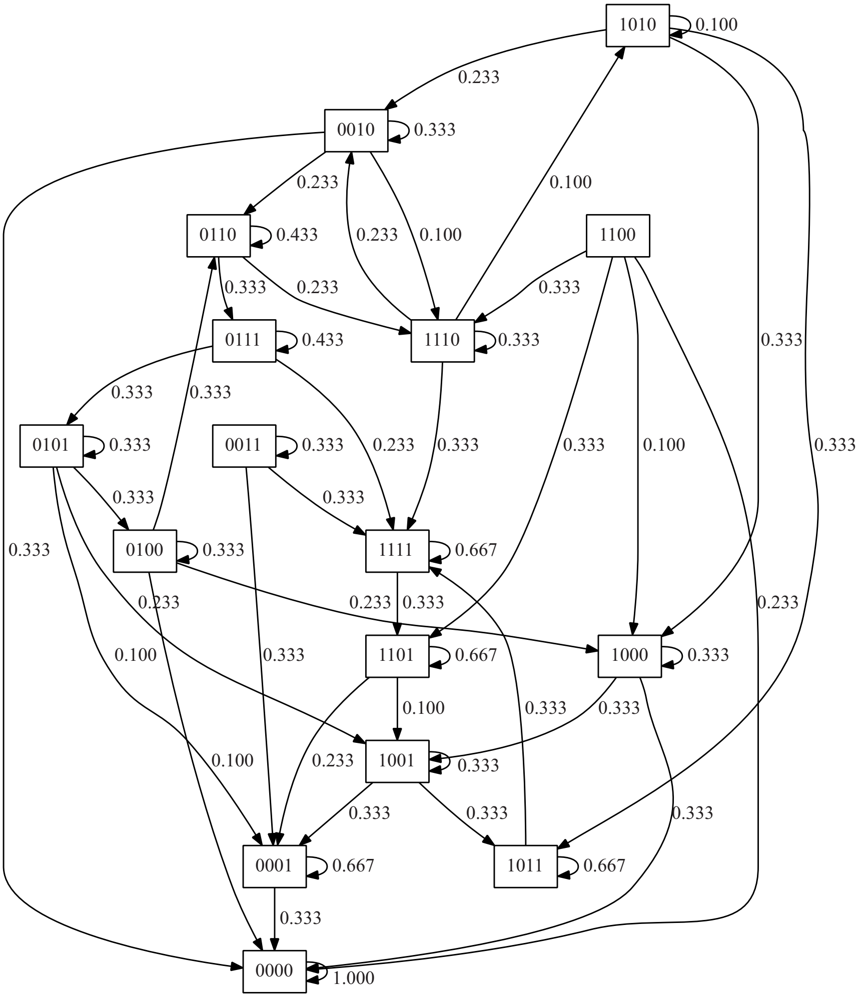


**Figure 38.7** Dynamical graph of a PBN with a mixed strategy considering $P = \{\{1, 2\}, \{3\}, \{4\}\}$.

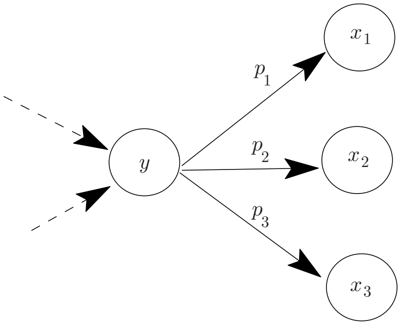


**Figure 38.8** Illustration of the Monte Carlo algorithms. The algorithm produces a random walk in the network in accordance with the probability distribution. Here, when one is in the state $y$, the choice to take the path through $x_1$, $x_2$, or $x_3$ is made in accordance with probabilities $p_1$, $p_2$, and $p_3$.

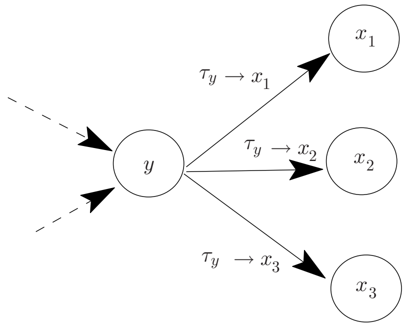


**Figure 38.9** Illustration of the Gillespie algorithms. The algorithm produces a random walk in the network in accordance with the production rates. Here, when one is in the state $y$, the choice to take the path through $x_1$, $x_2$, or $x_3$ is made in accordance with probabilities estimated from $\tau_{y \to x_1}$, $\tau_{y \to x_2}$, and $\tau_{y \to x_3}$.

The major issue of Monte Carlo approaches is to determine how many steps are necessary for an accurate estimation of the equilibrium. Moreover, biologists might be interested by the evolution of the quantities. It implies adding an estimation of time between two given quantities. In this purpose, the Gillespie algorithm [7] is a refinement of the Monte Carlo approach (see Figure 38.9). It uses a complementary information: the production rate $\tau$ for each interaction. The algorithm simulates the evolution of the biological compound over time by determining what kind, and when, the next interaction will occur. Like this, the simulation shows the result of a random walk in the discrete network for given initial conditions: the quantity of biological components in presence.

GILLESPIE ALGORITHM APPLIED ON PROBABILISTIC BOOLEAN NETWORKS. For a given quantity of all biological compounds of the system, and some production rates $\tau_{i \to j}$ from state $i$ to $j$ for all edges, the Gillepsie algorithm consists in a repetition of four basic steps. First suppose that the initial state is $i$.

1. Let $\tau_{\text{tot}} = \sum_j \tau_{i \to j}$.
2. Choose a random number $T$ following an exponential distribution with parameter $\tau_{\text{tot}}$. Here, $T$ is the total duration ellapsed in state $i$. Increase the total time by $T$.
3. Choose randomly the next state; each state $j$ is reached with probability $\tau_{i \to j} / \tau_{\text{tot}}$.
4. Update the production rates.

These simulations are particularly appropriate for estimating quantities of biological compounds that interact on large biological networks. The efficiency of the quantitative prediction can then be estimated using multiregression approaches. Note, herein, that automatic probabilistic verifications can be performed on smaller networks [9]. It emphasizes the impact of specific probabilities on the overall quantitative behaviors of the system. From the biological viewpoint, it indicates the genes, or other biological compounds, that can be tuned for a better fitting of the model with experimental data; or cornerstone genes that might impact the overall behavior when modified (*i.e.*, mutation or environmental condition modifications). The probabilistic model-checkers hence appear as a natural complement to the automatic qualitative verification techniques of the Boolean core, as previously mentioned in Section 38.1.1.

## 38.3 CONCLUSION

In this chapter, we summarized the essential features of the probabilistic Boolean networks. They represent a general probabilistic model that possesses plenty of applications in the context of biological networks when dedicated extensions are proposed. Notice that plenty other probabilistic models not shown herein exist. The Bayesian network is one of them that deals with biological informations. It is a probabilistic graph model that represents the biological compound interactions via a directed acyclic graph. As itself, it is not able to take into account the feedback loops. For taking them into account, one introduces dynamical Bayesian networks. They consist in a repetition of an elementary Bayesian network, as previously defined, that are linked together in order to abstract the dynamical effect, which includes the feedback loops. For further reading about this method as an extension of this chapter, we recommend the study [15] that compares the probabilistic Boolean networks and the dynamical Bayesian networks in a gene regulatory context.

## ACKNOWLEDGMENTS

The authors of this chapter would like to thank Mathieu Giraud and Pierre Peterlongo for their precious comments.

## REFERENCES

1. J. Ahmad, J. Bourdon, D. Eveillard, J. Fromentin, O. Roux, and C. Sinoquet. Temporal constraints of a gene regulatory network: Refining a qualitative simulation. *Biosystems*, 98(3):149–159, 2009.
2. C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Mach Learn*, 50:5–43, 2003.
3. J. Bourdon, D. Eveillard, S. Gabillard, and T. Merle. Using a probabilistic approach for integrating heterogeneous biological knowledges. *Proceedings of RIAMS 2007*, Lyon, 2007.
4. C. Chaouiya, H. de Jong, and D. Thieffry. Dynamical modeling of biological regulatory networks. *Biosystems*, 84(2):77–80, 2006.

5. H. de Jong. Modeling and simulation of genetic regulatory systems: A literature review. *J Comput Biol*, 9(1):67–103, 2002.
6. A. Fauré, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–131, 2006.
7. D.T. Gillespie. Stochastic simulations of coupled chemical reactions. *J Phys Chem*, 81(25):2340–2361, 1977.
8. K. Glass and S.A. Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *J Theor Biol*, 39:103–129, 1973.
9. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theor Comput Sci*, 391(3):239–257, 2008.
10. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA, 1979.
11. S. Huang. Gene expression profiling, genetic networks, and cellular states: An integrating concept for tumorigenesis and drug discovery. *J Mol Med*, 77(6):469–480, 1999.
12. S. Huang. Genomics, complexity and drug discovery: Insights from boolean network models of cellular regulation. *Pharmacogenomics*, 2(3):203–222, 2001.
13. S.A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J Theor Biol*, 22(3):437–467, 1969.
14. C.Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Syst Tech J*, 38:985–999, 1959.
15. P. Li, C. Zhang, E.J. Perkins, P. Gong, and Y. Deng. Comparison of probabilistic boolean network and dynamic Bayesian network approaches for inferring gene regulatory networks. *BMC Bioinformatics*, 8(Suppl 7):S13, 2007.
16. A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, 97(2):134–139, 2009.
17. A. Naldi, D. Thieffry, and C. Chaouiya. Decision diagrams for the representation and analysis of logical models of genetic networks. *Proceedings of the Computational Methods in Systems Biology (CMSB'07)*, volume LNCS/LNBI 4695, 2007, pp. 233–247.
18. D. Ropers, H. de Jong, M. Page, D. Schneider, and J. Geiselmann. Qualitative simulation of the carbon starvation response in Escherichia coli. *Biosystems*, 84(2):124–152, 2006.
19. I. Shmulevich, E.R. Dougherty, S. Kim, and W. Zhang. Probabilistic boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.
20. I. Shmulevich, E.R. Dougherty, and W. Zhang. Gene perturbation and intervention in probabilistic boolean networks. *Bioinformatics*, 18(10):1319–1331, 2002.
21. I. Shmulevich and W. Zhang. Binary analysis and optimization-based normalization of gene expression data. *Bioinformatics*, 18(4):555–565, 2002.
22. A. Siegel, O. Radulescu, M. Le Borgne, P. Veber, J. Ouy, and S. Lagarrigue. Qualitative analysis of the relation between DNA microarray data and behavioral models of regulation networks. *Biosystems*, 84(2):153–174, 2006.
23. D. Thieffry. Dynamical roles of biological regulatory circuits. *Brief Bioinformatics*, 8(4):220–225, 2007.

24. D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks—II. Immunity control in bacteriophage lambda. *Bull Math Biol*, 57(2):277–297, 1995.
25. D. Thieffry and R. Thomas. Qualitative analysis of gene networks. *Pacific Symposium on Biocomputing*, 1998, pp. 77–88.
26. R. Thomas, D. Thieffry, and M. Kaufman. Dynamical behaviour of biological regulatory networks—I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bull Math Biol*, 57(2):247–276, 1995.
27. P. Veber, C. Guziolowski, M. Le Borgne, O. Radulescu, and A. Siegel. Inferring the role of transcription factors in regulatory networks. *BMC Bioinformatics*, 9:228, 2008.

39

# MODELING AND ANALYSIS OF BIOLOGICAL NETWORKS WITH MODEL CHECKING

Dragan Bošnački, Peter A.J. Hilbers, Ronny S. Mans, and Erik P. de Vink

## 39.1 INTRODUCTION

During last decades, biological networks, like signal transduction pathways, metabolic pathways, and genetic networks, have received increasing attention in biochemistry. In each living organism, a growing plethora of such networks have been identified. It has become clear that the understanding of the mechanisms and their functioning is crucial for elucidating the functioning of the cell and the organism as a whole.

Different formalisms and approaches exist for the modeling of biological networks. In this chapter we focus on model checking as a method that exploits executable models. Its main advantage is that they lend themselves to formal verification. Standard simulation on the model can only yield predictions regarding model properties with certain probability. The advantage of model checking over standard simulation is that it considers all possible behaviors of the systems, not just some subset of it and therefore yields conclusions with certainty.

After intrducing the basic concepts in the next section, in Section 39.3, we show how standard model checking can be used to model and analyze biological systems. To this end we use as the modeling language Promela, the specification language of

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

the model checking tool SPIN. The SPIN tool can be used to check a broad range of properties. In particular, we show how SPIN can be used to detect steady states of the biological systems as well as periodic behavior. Some of the case studies that we discuss have also been modeled with other formalisms, like Petri nets or $\pi$-calculus. We discuss the advantages of model checking over those approaches.

Section 39.4 is devoted to modeling and analysis of biological systems that are inherently probabilistic. To this end we use a special kind of model checking—probabilistic model checking. We demonstrate the concepts of probabilistic model checking for biological systems using the probabilistic model checking tool Prism.

## 39.2 PRELIMINARIES

### 39.2.1 Model Checking

Roughly speaking, model checking [19, 2] is an automated technique that, given a model of the system and some property, checks whether the model satisfies the property. Compared with other automated or semi-automated formal techniques, such as deductive methods using theorem provers, model checking is relatively easy to use. The specification of the model is very similar to programming, and as such it does not require much additional expertise from the user. The verification procedure is completely automated and often takes only seconds to several minutes. Another important advantage of the method is that, if the verification fails (*i.e.*, the property that is checked does not hold), the erroneous behavior of the system can be reproduced. This significantly facilitates the location and correction of the errors.

Unlike simulation, model checking explores all possible states of the system. The model checker explores the complete system behavior (*i.e.*, all possible executions of the system). Obviously, for model checking to be applicable, the state space of the system under study should be finite. Systems with infinite state spaces can be handled as well provided the state space can be reduced to a finite one. For this reason, various abstraction techniques are available. In general, the state space that reflects all system behavior is represented as a graph in which the states are nodes and the edges are transitions between states. A particular behavior of the system, which we also refer to as an execution sequence or a path, can be represented by a path in the graph consisting of states and the transitions between them. Of particular interest are states from which there are no outgoing transitions, which are called deadlock states, as well as cyclic paths in the state-space graph. The deadlock states correspond to steady states in the real systems, whereas the cycles are associated with periodical behavior of the systems.

To illustrate the above notions related to the state spaces, we consider the simple genetic network given in Figure 39.1 a, which consists of three genes. Gene A is an inhibitor of gene B, whereas gene B activates gene C. Furthermore, if B is not active, then gene C spontaneously deactivates, whereas if gene A is not active, then gene B spontaneously activates. Also gene A is a self-activating one. The state space of the network is given in Figure 39.1b. The states are represented by a state vector. For

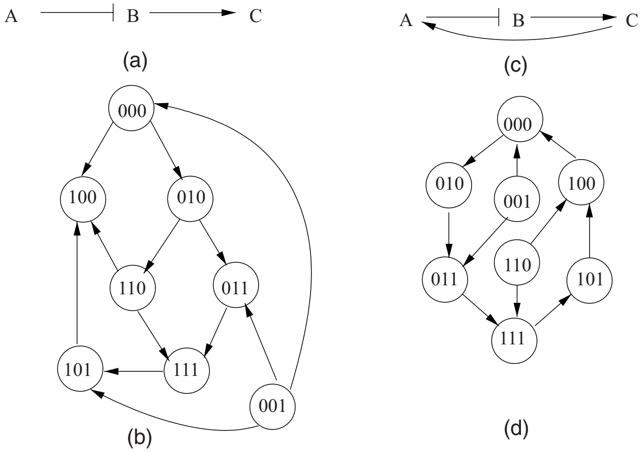


**Figure 39.1** A simple genetic network with its state space.

the sake of simplicity, in the figure, the state vectors are simplified into 3-bit binary strings. (States are depicted as circles, and transitions as arcs.) The components of the vector correspond to the genes A, B, and C, respectively. Each gene can take two values 0 (not activated) and 1 (activated).

We assume that the genes change their states asynchronously, one gene at a time. So, from the state 000 one can go nondeterministically either to state 100 or 010. The first transition happens when gene A activates spontaneously, whereas the second one corresponds to a spontaneous activation by gene B. The transition from state 101 to 100 happens by spontaneous deactivation of gene C and the transition from 110 to 100 because of the inhibiting influence of A to B. Similarly the other transitions can be coupled to activation/deactivation of a particular gene. The only deadlock or stable state is 100.

The state space does not contain cycles. However, cyclic behavior can be introduced, for instance, by adding a feedback activating influence between C and A and assuming that A deactivates spontaneously (see Figure 39.1c). In that case a spontaneously activated gene B activates gene C, which in its turn activates gene A. Gene A will deactivate gene B, which will result in spontaneous deactivation of gene C and as a consequence also gene A. Then the cycle can resume again with a new spontaneous activation of B. The state space of the new network is given in Figure 39.1d.

### 39.2.2 SPIN and Promela

Spin [18] is a software tool that supports the analysis and verification of concurrent systems. The system descriptions are modeled in a high-level language called Promela. Its syntax is derived from the programming language C and extended with constructs to model nondeterminism, the so-called guarded commands from Dijkstra, and with statements to model communication (sending and receiving messages) that are inspired from Hoare's CSP language.

In Promela, system components are specified as *processes* that can interact either by message passing, via *channels*, or memory sharing, via *global variables*. The message passing can either be buffered or unbuffered. Concurrency is asynchronous and modeled by interleaving (*i.e.*, in every step exactly one *enabled* action is performed, if available at all). No assumptions are made on the relative speed of process executions.

Given a Promela description as input, SPIN generates a C program that performs the verification of a system property by generating the state-space graph. Simultaneously, the check of the property is performed (*i.e.*, each new state is checked if it is erroneous, *e.g.*, a deadlock state, or if an erroneous cyclic path is closed). There are various ways to formally express the properties that we want to verify. Properties that boil down to the presence or absence of cycles in the state space can be formulated via special formal language called linear temporal logic (LTL) (*cf.* [12]). We give more details about LTL in Section 39.2.3. The most general way of expressing properties in SPIN is via so-called *never claims*, which are best seen as monitoring processes that run in lock step with the rest of the system.[1] SPIN provides an automatic translator from formulas in LTL to never claims. In case the system violates a property, the trace of actions leading to an invalid state, or a cycle, is reported. The erroneous trace can then be replayed, on the Promela source, by a guided simulation.

### 39.2.3 LTL

We give only an informal overview of the LTL. For a formal definition, we refer the reader to [12]. Temporal logic is a formalism for specifying sequences of states. Temporal logic formulae are composed out of a small number of special temporal operators and state formulae. LTL is a specific branch of temporal logic that only contains future time temporal operators. This branch of logic is most relevant to the verification of concurrent systems.

For our purposes we only use two temporal operators. These are the operator *always* or *box*, which is represented by the symbol "[ ]" and the operator *eventually* or *diamond*, which is represented by the symbol "$\langle\,\rangle$". Let us suppose that $p$ is a formula expressing some property. Then formula $[\,]p$ captures the notion that the property specified by $p$ remains invariantly *true* throughout an execution sequence (*i.e.*, holds in each state of the sequence). The informal meaning of the formula $\langle\,\rangle p$ is that the property $p$ is guaranteed to eventually become *true* at least once in an execution sequence. Besides the special temporal operators, LTL also provides the usual logical operators: "!" for negation, "||" for disjunction, "&&" for conjunction, and "$\rightarrow$" for logical implication.[2]

To illustrate the use of LTL formula, some examples are given in Table 39.1.

[1] The never claims are, in fact, Büchi Automata [33] and, thus, can express what are called arbitrary omega-regular properties.

[2] For the reader who is familiar with LTL and model checking: In most of the applications that we discuss in this chapter, the usage of LTL formulas even for safety properties (*i.e.*, properties that can be disproved with a finite counterexample sequence), is essential. This is because those safety properties are global and therefore cannot be expressed with assertions, which capture only local properties.

**Table 39.1 Examples of LTL formulas**

| Formula | Informal Meaning |
|---|---|
| $[\,]\langle\,\rangle p$ | always eventually $p$, i.e., infinitely many $p$'s |
| $\langle\,\rangle[\,]p$ | eventually always $p$, i.e., $p$ only from some point on |
| $p \rightarrow \langle\,\rangle q$ | if initially $p$ then eventually $q$ |
| $[\,](p \rightarrow \langle\,\rangle q)$ | every $p$ is eventually followed by a $q$ |
| $< > p \rightarrow \langle\,\rangle q$ | eventually $p$ implies eventually $q$ |

## 39.3 ANALYZING GENETIC NETWORKS WITH MODEL CHECKING

We discuss how to use model checking to analyze genetic networks. In genetic networks, genes can activate or inhibit one another. Moreover, self-regulation (activation/inhibition) of a gene is possible too. We are interested in the qualitative behavior of genetic networks, i.e., for each gene we distinguish only two possible states: 'on' and '1' vs. 'off' and '0'. Consequently, we use boolean regulatory graphs [9] as formal models of the genetic networks.

### 39.3.1 Boolean Regulatory Networks

Let $G = \{g_1, \ldots, g_n\}$ be a set of genes. To each gene $g_i \in G$, we assign a subset $I(i) \subseteq G$ and a boolean function $K_i$. Intuitively, $I(i)$ contains the source genes of all incoming interactions into $g_i$ and is called *input of* $g_i$. The boolean function $K_i\colon 2^{I(i)} \rightarrow \{0, 1\}$ associates a parameter $K_i(X)$ to each subset $X$ of $I(i)$. Intuitively, if all genes in the subset $X$ are active then $g_i$ is activated (if $K_i(X) = 1$) or inhibited (in case $K_i(X) = 0$). As a result, the output of function $K_i(X)$ produces the *new* value of gene $g_i$.[3] The corresponding regulatory graph is a (labeled) directed graph defined by the following three components:

- a set of nodes $G = \{g_1, \ldots, g_n\}$,
- a set of edges determined by the sets $I(i), i = 1, \ldots, n$, and
- a set of parameters $\mathcal{K} = \{K_i(X) \mid j = 1, \ldots, n, X \subseteq I(i)\}$.

### 39.3.2 A Case Study

We illustrate our approach in more detail on a case study—a genetic network of the plant *Arabidopsis thaliana*, which is involved in the control of flower morphogenesis. Mendoza *et al.* [27] have proposed a Boolean regulatory model involving 10 genes cross-regulating each other. For proper parameter value sets, this model encompasses six stable states, four of them matching the qualitative gene expression patterns observed in the different flower organs, whereas the two last stable states correspond to

[3] Thus, the network can be regarded as an asynchronous sequential logical circuit.

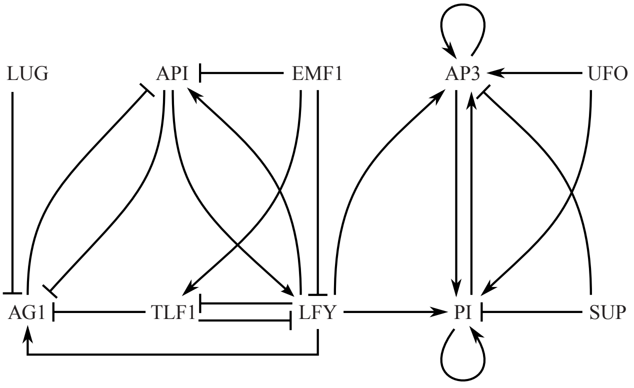


**Figure 39.2** Genetic network for flowering in *Arabidopsis*.

nonflowering situations. All these stable states correspond to deadlocks in the state space of the system and as such can be detected by model checking.

Chaouiya *et al.* have introduced a simplified version of the network that focuses on a subset of six genes that play a crucial role in the selection of specific flowering differentiative pathways, leaving aside the genes that can be treated as simple inputs (EMF1, UFO, LUG, and SUP). In [9], a parameter set has been chosen for which the system has four stable states, each corresponding to a gene expression pattern associated with a specific flower organ.

Thus, only the following genes are considered: TLF1, LFY, AP1, AG1, AP3, and PI. The gene network is depicted in Figure 39.2. Crossbar arrowheads indicate inhibition, whereas standard arrowheads indicate activation. The inhibition/activation interactions between genes in Figure 39.2 are rather informal. The precise definition of the interactions is given by the $K_i$ parameters of the underlying regulatory graph that are given in Table 39.2. In the table $g_i$ is associated with $g_i = 1$ $(g_i \in X)$ and its complement $\bar{g}_i$ corresponds to $g_i = 0$ $(g_i \notin X)$. For example, $K_L(A\bar{T}) = 0$ means that when AP1 is activated and TLF1 is inhibited, then gene LFY is inhibited.

**Table 39.2 Parameters given in [9]**

| TLF1 (=T) | LFY (=L) | AP1 (=A) | AG1 (=G) | AP3 (=P) | PI (=I) |
|---|---|---|---|---|---|
| $K_T(\bar{L}) = 0$ | $K_L(\bar{A}\bar{T}) = 0$ | $K_A(\bar{L}\bar{G}) = 1$ | $K_G(\bar{T}\bar{L}\bar{A}) = 1$ | $K_P(\bar{P}\bar{I}\bar{L}) = 0$ | $K_I(\bar{I}\bar{P}\bar{L}) = 0$ |
| $K_T(L) = 0$ | $K_L(\bar{A}T) = 0$ | $K_A(\bar{L}G) = 0$ | $K_G(\bar{T}\bar{L}A) = 0$ | $K_P(\bar{P}\bar{I}L) = 1$ | $K_I(\bar{I}\bar{P}L) = 1$ |
| | $K_L(A\bar{T}) = 0$ | $K_A(L\bar{G}) = 1$ | $K_G(\bar{T}L\bar{A}) = 1$ | $K_P(\bar{P}I\bar{L}) = 0$ | $K_I(\bar{I}P\bar{L}) = 0$ |
| | $K_L(AT) = 0$ | $K_A(LG) = 0$ | $K_G(\bar{T}LA) = 0$ | $K_P(\bar{P}IL) = 1$ | $K_I(\bar{I}PL) = 1$ |
| | | | $K_G(T\bar{L}\bar{A}) = 0$ | $K_P(P\bar{I}\bar{L}) = 0$ | $K_I(I\bar{P}\bar{L}) = 0$ |
| | | | $K_G(T\bar{L}A) = 0$ | $K_P(P\bar{I}L) = 0$ | $K_I(I\bar{P}L) = 0$ |
| | | | $K_G(TL\bar{A}) = 0$ | $K_P(PI\bar{L}) = 1$ | $K_I(IP\bar{L}) = 1$ |
| | | | $K_G(TLA) = 0$ | $K_P(PIL) = 1$ | $K_I(IPL) = 1$ |

### 39.3.3 Translating Boolean Regulatory Graphs into Promela

As a Boolean function, each $K_i$ parameter has a unique value for all possible combinations of inputs. Therefore, it is relatively straightforward to model such functions in Promela. We model each gene $g_i$ as a separate process (Promela proctype), which consists of an (infinite) loop given with the Promela do loop:

```
do
:: statement_1
:: statement_2
...
:: statement_n
od.
```

Each `statement` is of the form `condition->action` with the meaning that `action` is executed if the guard `condition` is fulfilled. Furthermore, each `statement` corresponds to a row in definition of function $K_i$ in Table 39.2. For instance, the code for the AG1 gene is given in Listing 39.3.1.

Listing 39.3.1 (`Promela code corresponding to gene AG1`)

```
proctype AG() {
do
:: atomic{!Active[G] && !Active[T] && !Active[L] && !Active[A] -> Active[G]=1}
:: atomic{!Active[G] && !Active[T] &&  Active[L] && !Active[A] -> Active[G]=1}

:: atomic{Active[G] && !Active[T] && !Active[L] &&  Active[A] -> Active[G]=0}
:: atomic{Active[G] && !Active[T] &&  Active[L] &&  Active[A] -> Active[G]=0}
:: atomic{Active[G] &&  Active[T] && !Active[L] && !Active[A] -> Active[G]=0}
:: atomic{Active[G] &&  Active[T] && !Active[L] &&  Active[A] -> Active[G]=0}
:: atomic{Active[G] &&  Active[T] &&  Active[L] && !Active[A] -> Active[G]=0}
:: atomic{Active[G] &&  Active[T] &&  Active[L] &&  Active[A] -> Active[G]=0}
od; }
```

Recall that the exclamation mark `!` denotes negation. The `atomic` clause is a technicality that denotes that the enclosed check of the guard and the corresponding action are executed in atomic fashion (*i.e.*, they cannot be interrupted by some statement executed by another process (gene)). The complete model is given in the appendix as Listing A.0.1.

The do loop is executed as long as at least one of the options is executable. Otherwise the loop is blocked. Each branch (guarded command) of the do loop corresponds to a row in the table defining $K_i$. The condition (guard) encodes the fact that all genes in $X$ are active. As a result, the value of $g_i$ is updated according to $K_i$ from the table.

We simplify the model by the observation that to detect stable states, only the transition of a gene from active to inactive and inactive to active needs to be considered. (When we also model a transition of a gene from active to active and from inactive to inactive, then the system can do a step, whereas the state of the system remains the same. As a result, a deadlock can never appear in the system.) Therefore, an extra variable $x_0$ is added to $K_i(X)$, which represents the gene $i$ itself, and with which only transitions from inactive to active and from active to inactive can be considered.

Using fairly standard techniques from propositional logic, one can also simplify the Boolean functions. This often leads to a more compact code, which can be particularly useful for more complex networks. For example, the Promela code of the simplified logical expression of gene AG1 is presented in Listing 39.3.2.

Listing 39.3.2 (Simplified code for gene AG1)

```
proctype AG() {
do
:: atomic{!Active[G] && (!Active[T] && !Active[A]) -> Active[G]=1}
:: atomic{ Active[G] &&  (Active[T] ||  Active[A]) -> Active[G]=0}
od; }
```

### 39.3.4 Some Results

*Finding Stable States.* With the kind of models described above, one can find stable states by checking for deadlocks that in SPIN are called *invalid end states*. Since a deadlock state is an error SPIN also always shows a scenario that leads to the found deadlock state. Finding deadlock states in SPIN is an option that is independent from the LTL verification. By default SPIN stops after the first deadlock state is found. This is not very convenient because in this way it is possible to detect only one stable state in the model. In principle, there is an option that instructs SPIN not to stop on the first error—in our case, the first found deadlock—and instead report all found deadlocks. However, for technical reasons that are beyond the scope of this chapter and that are related to SPIN's output, sometimes it could be more convenient to use the following trick: To each found stable state, we add a self-loop to that sate. In this way, the latter is not found anymore by the deadlock detection algorithm. To achieve this, we add a separate process with a `do` loop in it that contains the following line: `stable_state -> skip` (where `stable_state` is a correct representation of the stable state in SPIN, and `skip` is a dummy statement). In this way, there is always a transition from `stable_state` to itself. Obviously, one could repeat this procedure until no more stable states are found.

As it was mentioned above, to simplify the model and make the verification more efficient, we exploited the fact that in our setting the stable states correspond to deadlock states. However, with model checking techniques, one can detect also a more general type of stable states that are not necessarily deadlocks. They correspond to partially stable states in which only a subset of the genes in the network remains stable.

$$\begin{bmatrix} T=0 \\ L=0 \\ A=1 \\ G=0 \\ P=0 \\ I=0 \end{bmatrix} \begin{bmatrix} T=0 \\ L=0 \\ A=1 \\ G=0 \\ P=1 \\ I=1 \end{bmatrix} \begin{bmatrix} T=0 \\ L=0 \\ A=0 \\ G=1 \\ P=0 \\ I=0 \end{bmatrix} \begin{bmatrix} T=0 \\ L=0 \\ A=0 \\ G=1 \\ P=1 \\ I=1 \end{bmatrix}$$

**Figure 39.3** The four stable states.

Finding such states can be achieved in two steps. First, we have to verify that the state itself can be reached. This can be done with the following formula (that, by default, is checked for all executions): ! ⟨⟩ `state` meaning "not eventually `state`" (*i.e.*, "`state` is never reachable". When the model checker gives an error trail, then the state exists. Second, we have to verify that the state is indeed a stable state. In other words, when the system reaches the state, it has to remain in that state forever. This can be done with the following formula, which has to hold for all executions of the system: `[](state -> []state)`. When the model checker indicates that the verification result is valid, then `state` is indeed stable.

Using deadlock detection techniques, we obtain some interesting results with the *Arabidopsis* model. In [9] it is claimed that, for the set of parameters given in Table 39.2 and with an initial state in which both LFY and AP1 are active and the others being not active, the system has four stable states. These stable states are shown in Figure 39.3. (The first stable state in the figure indicates that gene A (AP1) is active and the other genes are not. The others can be interpreted analogously.) However, SPIN reports that the last two stable states in Figure 39.3 cannot be reached at all. This can be also analytically proven.

Guided by the counterexamples produced by SPIN, we define an alternative set of parameter values given in Table 39.3 for which all four stable states exist. When choosing these parameter values, we have tried to respect as much as possible the activatory and inhibitory relationships among the genes, as defined in Figure 39.2. Sometimes this was impossible though. So, in the cases in which the pictorial representation is ambiguous (*i.e.*, for the genes for which there are both activating and inhibiting incoming edges), we have used the predetermined values of the

**Table 39.3 Set of parameters that respect as much the activatory and repressory relationships and the predetermined values**

| TLF1 (=T) | LFY (=L) | AP1 (=A) | AG1 (=G) | AP3 (=P) | PI (=I) |
|---|---|---|---|---|---|
| $K_T(\bar{L}) = 0$ | $K_L(\bar{A}\bar{T}) = 0$ | $K_A(\bar{L}\bar{G}) = 1$ | $K_G(\bar{T}\bar{L}\bar{A}) = 1$ | $K_P(\bar{P}\bar{I}\bar{L}) = 0$ | $K_I(\bar{I}\bar{P}\bar{L}) = 0$ |
| $K_T(L) = 0$ | $K_L(\bar{A}T) = 0$ | $K_A(\bar{L}G) = 0$ | $K_G(\bar{T}\bar{L}A) = 0$ | $K_P(\bar{P}\bar{I}L) = 1$ | $K_I(\bar{I}\bar{P}L) = 1$ |
| | $K_L(A\bar{T}) = 0$ | $K_A(L\bar{G}) = 1$ | $K_G(\bar{T}L\bar{A}) = 1$ | $K_P(\bar{P}I\bar{L}) = 1$ | $K_I(\bar{I}P\bar{L}) = 1$ |
| | $K_L(AT) = 0$ | $K_A(LG) = 1$ | $K_G(\bar{T}LA) = 1$ | $K_P(\bar{P}IL) = 1$ | $K_I(\bar{I}PL) = 1$ |
| | | | $K_G(T\bar{L}\bar{A}) = 0$ | $K_P(P\bar{I}\bar{L}) = 1$ | $K_I(I\bar{P}\bar{L}) = 1$ |
| | | | $K_G(T\bar{L}A) = 0$ | $K_P(P\bar{I}L) = 1$ | $K_I(I\bar{P}L) = 1$ |
| | | | $K_G(TL\bar{A}) = 1$ | $K_P(PI\bar{L}) = 1$ | $K_I(IP\bar{L}) = 1$ |
| | | | $K_G(TLA) = 1$ | $K_P(PIL) = 1$ | $K_I(IPL) = 1$ |

parameters, as given in [27]. That those predetermined values are not always in accord with Figure 39.2 can be seen from the following example. In [27], it has been stated that in order to obtain stable states $K_L(AT)$ needs to be set to zero. Since we have that in two of the stable states AP1 is activated and LFY is inhibited, we conclude that $K_L(A)$ has to be set to zero too. This is contradictory with the relationships among the genes in Figure 39.2, which imply that gene AP1 activates gene LFY. By applying similar reasoning, we were able to establish the parameter set given in in Table 39.3. With these values for all states in Figure 39.3, we could check with SPIN that they can be reached and that they are indeed stable states. It should be emphasized that despite some discrepancies, the values from Table 39.3 are much closer to the experimental data (see e.g., [27]) than the parameters in Table 39.2 proposed in [9]. Thus, in this way we are able to substantially improve the model based on the results obtained with the model checker.

*Checking Cycles.* Besides stable states, one can also detect cycles in the state space of the network. Suppose that we want to check if there is a cycle consisting of (a subset of) the states $s_0$ to $s_n$. To find the cycle we have to check the negation of the formula: $!<>[\,](s_0||s_1||\ldots||s_n)$ (*i.e.*, not eventually always one of the states $s_0$ to $s_n$). In other words, there is no execution sequence of the model such that some of the states $s_0$ to $s_n$ is reached, and the system remains in that state forever. If the cycle exists, the previous property does not hold. Strictly speaking, one has to check first that none of the states $s_i$, $i = 0 \ldots n$ is a deadlock state. If one of the states turns out to be a deadlock, this would automatically mean that a cycle through composed of those states does not exist.

It is worth emphasizing that it takes SPIN just a couple of seconds to produce all results described above.

### 39.3.5 Concluding Remarks

In the previous sections, we considered genetic networks that are defined by a Boolean function. We presented a method for which a genetic network, which is defined by a Boolean function can be translated into Promela. With a small extension to this method, it is possible to detect stable states in the network. Also, we showed some LTL formulas with which it is possible to verify stable states and cycles.

The use of standard model checking techniques, exploiting Promela and SPIN in particular, are not limited to genetic networks. They can be applied also to other types of biological networks, like signaling and metabolic pathways. More specifically, since SPIN originally has been developed for modeling of communication protocols, it can be used in a natural way for modeling signaling pathways.

### 39.3.6 Related Work and Bibliographic Notes

There are several papers on model checking or closely related formal techniques applied to biological networks (*e.g.*, [20, 6, 30, 25]). Here we briefly discuss in more detail the works that are the most relevant with regard to the subjects treated in this chapter.

In [8], symbolic model checking techniques are applied to the querying and validation of both quantitative and qualitative models of biomolecular systems. The main difference with our approach is that [8] uses the symbolic model checker NuSMV and the constraint-based model checker DMC, which both accept the temporal logic CTL as a language to formulate the queries (properties). It is well known that for some applications the explicit model checkers, like SPIN, are more efficient and more intuitive to use. For instance, it would be easier to translate the $\pi$-calculus into SPIN than into NuSMV. Also the property language of SPIN, LTL, and CTL are not comparable in the sense that there are LTL formulas that cannot be expressed in CTL and vice versa.

In [3] an approach for model checking genetic regulatory networks has been proposed that consists in connecting GNA to the CADP verification toolbox. GNA is a quantitative simulation tool well adapted to the available information on genetic regulatory networks. Also, it is capable of analyzing large and complex genetic regulatory networks. The $\mu$-calculus has been used as a property language. Although the $\mu$-calculus is more general than LTL, in the sense that each LTL formula there is an equivalent $\mu$-calculus formula, the latter are usually much more cryptic and difficult to grasp than their LTL counterparts.

In [1], two tools are described, Simpathica and XSSYS, which involves an automaton-based semantics of the temporal evolution of complex biochemical reactions starting from the representation given as a set of differential equations. Also, the ability is provided to qualitatively reason about the systems using a propositional temporal logic. However, those tools are essentially simulation tools that deal with systems that are deterministic with nature. The Promela/SPIN models are able to capture also the nondeterministic feature of the biological systems.

SPIN is an open-source software that is available from `http://spinroot.com`. Other model checking tools can be applied also to biological systems, for instance, NuSMV available from `http://nusmv.irst.itc.it`, and DiVinE, available from `http://divine.fi.muni.cz`.

## 39.4 PROBABILISTIC MODEL CHECKING FOR BIOLOGICAL SYSTEMS

Many biological systems have inherently a probabilistic/stochastic nature. A probabilistic interpretation, rather than a deterministic one underlying the continuous view based on ordinary differential equations (ODEs), is necessary when the number of molecules in the system is small or the time interval considered is short. A standard example from the literature are genetic switches, in particular the $\lambda$-phage [26]. In this section we consider another class of biological systems in which probabilities play an indispensable role. We show how a specific type of model checking, so-called probabilistic model checking, can be used for such systems. The probabilistic approach we present has been developed in the last several years and constitutes an alternative to traditional methods such as Gillespie-type simulations. Also here, the advantage of probabilistic model checking over simulation is that model checking considers all possible behaviors of the systems (*i.e.*, all simulation runs). Thus, model checking, when feasible, is more reliable and in the case considered also faster than simulation. To demonstrate the basic approach, we use the probabilistic model checker Prism.

### 39.4.1 Motivation and Background

The transfer of genetic information from DNA to mRNA to protein happens with very high precision. This is because each a single error potentially can have dramatic consequences for the organism as a whole. Here we analyze the stage of this information pathway that corresponds to the translation from mRNA to protein (*i.e.*, the protein biosynthesis). In particular, we are interested in translation errors and the factors of potential influence.

An mRNA molecule can be considered as a string of codons, each of which encodes for a specific amino acid. The codons of an mRNA molecule are sequentially read by a ribosome, and each codon is translated into an amino acid. As a result we obtain a chain of amino acids (*i.e.*, a protein). The amino acids are carried to the ribosome by a specific transfer-RNA (aa-tRNA). Each aa-tRNA contains a so-called anticodon and carries a specific amino acid. Arriving by Brownian motion, an aa-tRNA docks into the ribosome and may succeed in adding its amino acid to the chain under construction. Alternatively, the aa-tRNA dissociates in some stage of the translation. This depends on the pairing of the codon under translation with the anticodon of the aa-tRNA, as well as on the stochastic influences such as the changes in the conformation of the ribosome.

Thanks to the vast amount of research during the last 30 years, the overall process of translation is reasonably well understood from a qualitative perspective. The process can be divided into around 20 small steps/reactions, with several of them being reversible. Relatively little is known exactly about the kinetics of the translation. During the past several years, Rodnina and collaborators have measured kinetic rates for various steps in the translation process for a small number of specific codons and anticodons [28, 31, 32, 15]. They were able to experimentally show that in several of those steps, the rates strongly depend on the degree of matching between the codon and the anticodon. Additionally, in [11], the average concentrations (amounts) of aa-tRNAs per cell have been collected for the model organism *Escherichia coli*. Viljoen and co-workers [13] proposed a model that is based on those results. One of their basic assumption is that the rates found by Rodnina *et al.* can be used, in general, for all codon-anticodon pairs. Thus, the model in [13] covers all 64 codons and all 48 aa-tRNA classes for *E. coli*. The model is used to perform extensive Monte Carlo simulations and to establish codon insertion times and frequencies of erroneous elongations. The results show a strong correlation of the translation error and the ratio of the concentrations of the so-called near-cognate and cognate aa-tRNA species. Consequently, one can argue that the competition of aa-tRNAs, rather than their availability, decides both speed and fidelity of codon translation.

In this text, we model the translation kinetics via the modelchecking of continuous-time Markov chains (CTMCs) using the tool Prism [22, 16]. The tool provides built-in performance analysis algorithms and a formalism (computational stochastic logic, CSL) to reason about various properties of the CTMCs, removing the burden of extensive mathematical calculations from the user. Additionally, in our case, the Prism tool provides much shorter response times compared with Gillespie simulation.

### 39.4.2 A Kinetic Model of mRNA Translation

There exists a fixed correspondence between codons and amino acids given by the well-known genetic code. With exception of the three so-called stop codons, which denote the end of the genetic message, each codon codes for exactly one amino acid. As a consequence, an mRNA encodes for a unique protein. This ideal picture is disturbed by the fact that, in theory, each codon can bind with each anti-codon. However, the binding intensity can significantly differ from pair to pair. This influences the speed of the actual translation and the chances for errors. Thus, the translation is accurate but not perfect. The biological model of the translation mechanism that we adopt here is based on [31, 21]. Two main phases can be distinguished: peptidyl transfer and translocation. Here we focus on the peptidyl transfer because it is this part that determines the error probabilities. This phase can be divided in several steps that are represented in Figure 39.4 and that we briefly describe in the sequel. The transfer begins with aa-tRNA arriving at the A-site of the ribosome-mRNA complex by diffusion (state A1 in Figure 39.4). The initial binding leads

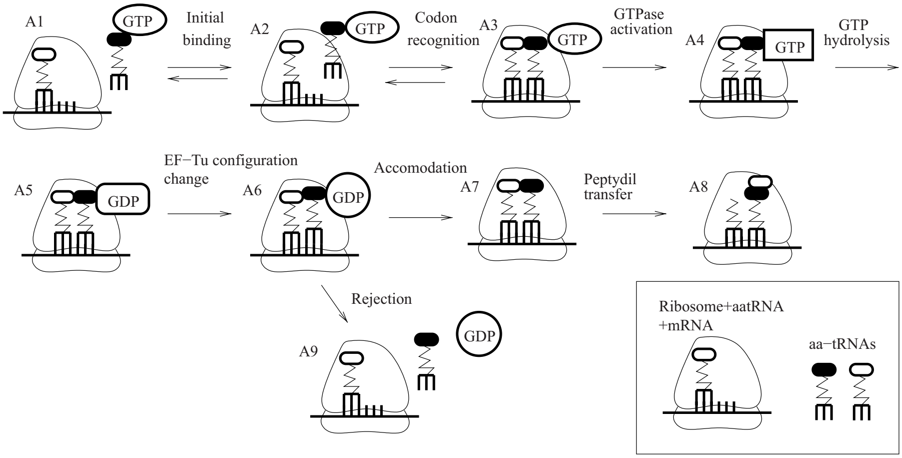


**Figure 39.4** Kinetic scheme of peptidyl transfer [13].

to state A2. As the binding is relatively weak the reverse process (*i.e.*, unbinding of the aa-tRNA is also possible), this brings us to the initial state. Codon recognition comprises (i) establishing contact between the anticodon of the aa-tRNA and the current codon in the ribosome-mRNA complex, and (ii) subsequent conformational changes of the ribosome. *GTP*ase-activation of the elongation factor *EF-Tu* is largely favored in case of a strong complementary matching of the codon and anticodon. After *GTP*-hydrolysis, producing inorganic phosphate $P_i$ and *GDP*, the affinity of the ribosome for the aa-tRNA reduces. The subsequent accommodation step also depends on the fit of the aa-tRNA. This step happens rapidly for cognate *aa-tRNA*, whereas for near-cognate *aa-tRNA*, this proceeds slower and the *aa-tRNA* is likely to be rejected. These different speeds are expressed via the reaction rates from A6 to A7 and A6 to A9. For a cognate *aa-tRNA*, the rate A6–A7 is much bigger than the rate A6–A9, whereas for a near cognate, the situation is the other way around.

Next, the translocation phase follows. Another GTP-hydrolysis involving elongation factor *EF-G* produces *GDP* and $P_i$ and results in unlocking and movement of the aa-tRNA to the P-site of the ribosome. The latter step is preceded or followed by $P_i$-release. Reconformation of the ribosome and release of *EF-G* moves the tRNA, which has transferred its amino acid to the polypeptide chain, into the so-called E-site of the ribosome. Further rotation eventually leads to dissociation of the used tRNA. As mentioned above, we assume that this phase does not further influence the probability of incorporating the amino acid in the chain. More precisely, we assume that once the final state (A8) of the peptidyl transfer is reached, the amino acid will be for sure added to the chain. Because of that in our formal model that we present later, we deal only with the peptidyl transfer.

There is not much quantitative information regarding the translation mechanism. For *E. coli*, several specific rates have been collected [31, 15], whereas some steps are known to be relatively rapid. Here we adopt the fundamental assumption of [13] that the experimental data found by Rodnina *et al.* for the UUU and CUC codons, extrapolate to other codons as well. Also, accurate rates for the translocation phase are largely missing. Again following [13], we have chosen to assign, if necessary, high rates to steps for which data are lacking. This way these steps will not be rate limiting.

### 39.4.3 Probabilistic Model Checking

Traditional model checking, which we presented in the previous sections of this chapter, deals with the notion of absolute correctness or failure of a given property. On the other hand, probabilistic[4] model checking is motivated by the fact that

[4] In the literature, probabilistic and stochastic model checking often are used interchangeably. A more clear distinction is made by relating the adjectives probabilistic and stochastic to the underlying model, viz. discrete-time and continuous-time Markov chain, respectively. For the sake of simplicity, in this chapter, our focus is on discrete-time Markov chains, so we opted for consistently using the qualification "probabilistic." Nevertheless, as we also emphasize in the sequel, the concepts and algorithms that we present here can be applied as well to continuous-time Markov chains.

probabilities are often an unavoidable ingredient of the systems we analyze. Therefore, the satisfaction of properties is quantified by a probability. This makes probabilistic model checking a powerful framework for modeling various systems ranging from randomized algorithms via performance analysis to biological networks.

From an algorithmic point of view, probabilistic model checking overlaps with the conventional technique, since it too requires computing reachability of the underlying state space graphs. Still, there are also important differences because numerical methods are used to compute the transition probabilities. However, these details are beyond the scope of this chapter. They will play no role in the application to biological systems that we consider in the sequel.

### 39.4.4 The Prism Model

To obtain our formal Prism model, we apply a twofold abstraction to the above informally sketched biological model: (i) Instead of dealing with 48 classes of aa-tRNA, that are identified by the their anticodons, we use four types of aa-tRNA distinguished by their matching strength with the codon under translation. (ii) We combine various detailed steps into one transition. The first reduction greatly simplifies the model, more clearly eliciting the essentials of the underlying process. The second abstraction is more a matter of convenience, although it helps in compactly presenting the model.

For each codon, we distinguish four types of aa-tRNA: *cognate, pseudo-cognate, near-cognate, and non-cognate*. Cognate aa-tRNAs carry an amino acid that is the correct one for the according to the genetic code and their anticodon strongly couples with the codon. The binding of the anticodon of a pseudo-cognate aa-tRNA or a near-cognate aa-tRNA is weaker, but sufficiently strong to occasionally result in the addition of the amino acid to the nascent protein. In case the amino acid of the aa-tRNA is, accidentally, the right one for the codon, we call the aa-tRNA of the pseudo-cognate type. If the amino acid does not coincide with the amino acid the codon codes for, we speak in such a case of a near-cognate aa-tRNA.[5] The match of the codon and the anticodon can be very poor too. We refer to such aa-tRNA as being non-cognate for the codon. This type of aa-tRNA does not initiate a translation step at the ribosome.

Here we focus on the computation of insertion errors. As a result the model can be even further simplified by assuming that the non-cognates do not play any role in the process. In our model, the main difference of cognates versus pseudo-cognates and near-cognates is in the kinetics. At various stages of the peptidyl transfer, the rates for true cognates differ from the others up to three orders of magnitude.

Figure 39.5 depicts the relevant abstract automaton, derived from the Prism model discussed above. In case a transition is labeled with two rates, the left-hand number concerns the processing of a cognate aa-tRNA, and the right-hand number that of

[5] The notion of a pseudo-cognate comes natural in our modeling. However, the distinction between a pseudo-cognate and a near-cognate is nonstandard. Usually, a near-cognate refers to both type of tRNA.

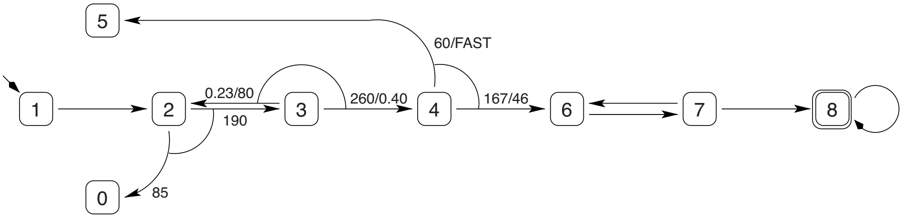


**Figure 39.5** Abstract automaton for error insertion.

a pseudo-cognate or near-cognate. In three states, a probabilistic choice has to be made. The probabilistic choice in state 2 is the same for cognates, pseudo-cognates, and near-cognates alike; the ones in state 3 and in state 4 differ for cognates and pseudo-cognates or near-cognates.

For example, after recognition in state 3, a cognate aa-tRNA will go through the hydrolysis phase leading to state 4 for a fraction 0.999 of the cases (computed as $260/(0.23 + 260)$), a fraction being close to 1. In contrast, for a pseudo-cognate or near-cognate aa-tRNA, this is 0.005 only. Cognates will accommodate and continue to state 6 with probability 0.736, whereas pseudo-cognates and near-cognates will do so with the small probability 0.044, the constant FAST being set to 1000 in our experiments. As the transition from state 4 to state 6 is irreversible, the rates of the remaining transitions are not of importance here.

One can see the Prism model as a superposition of three stochastic automata, each encoding the interaction of one of the types of aa-tRNA, except the non-cognate type. Each automaton is obtained from the automaton in Figure 39.5 by applying the corresponding rates.

We can further simplify our model by taking into account that we deal with average transition times and probabilities based on exponential distributions. Under this assumption, it is a common practice in performance analysis to merge two subsequent sequential transitions with given rates $\lambda$ and $\mu$ into a combined transition of rate $\lambda\mu/(\lambda + \mu)$. However, it should be noted that in general such a simplification is not compositional and should be taken with care.

In the models that we are considering, which are based on continuous-time Markov chains, Prism commands have the form `[label] guard` → `rate`: `update` ;. From the commands whose guards are satisfied in the current state, one command is selected with a probability proportional to its relative rate. Thus, a probabilistic choice is made. Executing the selected command results in a progress of time according to the exponential distribution for the particular rate. Also an update is performed on the state variables. More information about the Prism model checker can be found in [22, 16].

Initially, control is in state `s=1` of the Prism model with four Boolean variables `cogn`, `pseu`, `near`, and `nonc` set to false. The initial binding of aa-tRNA is modeled by selecting one of the Boolean variables that is to be set to true. There is a race among the three types of aa-tRNA: cognate, pseudo-cognate, or near-cognate.

The outcome of the race depends on the concentrations c_cogn, c_pseu, c_near, and c_nonc of the three types of aa-tRNA and a kinetic constant k1f. According to the Markovian semantics, the probability that cogn is set to true (the others remaining false) is the relative concentration c_cogn/(c_cogn + c_pseu + c_near). Analogously the probabilities for the other two types of aa-tRNA are computed. This amounts to the following code:

```
// initial binding
[ ] (s=1) -> k1f * c_cogn : (s'=2) & (cogn'=true) ;
[ ] (s=1) -> k1f * c_pseu : (s'=2) & (pseu'=true) ;
[ ] (s=1) -> k1f * c_near : (s'=2) & (near'=true) ;
```

The aa-tRNA that has just attached can also dissociate. We model this below by returning the control to the state s=0. Although it might seem more natural to return to the initial state, as we will see later, we need this state for model checking purposes. The Boolean that was set to true is reset. We assume the same dissociation rate for all aa-tRNA types. rate k2b.

```
// dissociation
[ ] (s=2) -> k2b ;
        (s'=0) & (cogn'=false) & (pseu'=false) & (near'=false) ;
```

Regardless of the type, aa-tRNA can continue from state s=2 in the codon recognition phase, leading to state s=3. This step can also be reversed; hence, we include transitions from state s=3 back to state s=2. The fidelity of the translation mechanism is ensured by the fact that the rates for cognates versus pseudo- and near-cognates, viz. k3bc, k3bp, and k3bn, differ significantly (see Table 39.4). The Boolean variables remain unchanged because aa-tRNA is not released.

```
// codon recognition
[ ] (s=2)        -> k2f :  (s'=3) ;
[ ] (s=3) & cogn -> k3bc : (s'=2) ;
[ ] (s=3) & pseu -> k3bp : (s'=2) ;
[ ] (s=3) & near -> k3bn : (s'=2) ;
```

The next step, from state s=3 to state s=4, is one-direcitonal. It corresponds to a combination of detailed steps in the biological model that involves modification of GTP. We assume that rates k3fp and k3fn, respectively, for pseudo-cognate and

**Table 39.4 Rates of the Prism model**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| k1f | 140 | k3fc | 260 | k4rc | 60 | k6f | 150 |
| k2f | 190 | k3fp, k3fn | 0.40 | k4rp, k4rn | FAST | k7f | 145.8 |
| k2b | 85 | k3bc | 0.23 | k4fc | 166.7 | k7b | 140 |
| k2bx | 2000 | k3bp, k3bn | 80 | k4fp, k4fn | 46.1 | | |

near-cognate aa-tRNA, are equal. The progress of the translation in the right direction is again ensured by a significant difference between these rates and rate `k3fc` for a cognate aa-tRNA.

```
// GTPase activation, GTP hydrolysis, EF-Tu conformation change
[ ] (s=3) & cogn -> k3fc : (s'=4) ;
[ ] (s=3) & pseu -> k3fp : (s'=4) ;
[ ] (s=3) & near -> k3fn : (s'=4) ;
```

State `s=4` is an important crossroad in the process. The aa-tRNA can either be rejected, after which control moves to the state `s=5`, or it can be accepted. This corresponds to the various accommodation steps in the biological model (*i.e.*, the ribosome reconforms such that the aa-tRNA can hand over the amino acid it carries), the so-called peptidyl transfer. In our model the accepting step means moving to to state `s=6`. In this step too the rates for cognates and those for pseudo-cognates and near-cognates differ significantly.

```
// rejection
[ ] (s=4) & cogn -> k4rc : (s'=5) & (cogn'=false) ;
[ ] (s=4) & pseu -> k4rp : (s'=5) & (pseu'=false) ;
[ ] (s=4) & near -> k4rn : (s'=5) & (near'=false) ;
// accommodation, peptidyl transfer
[ ] (s=4) & cogn -> k4fc : (s'=6) ;
[ ] (s=4) & pseu -> k4fp : (s'=6) ;
[ ] (s=4) & near -> k4fn : (s'=6) ;
```

The step from state `s=6` to state `s=7` models the binding of the EF-G complex. This step is also reversible, but eventually the binding becomes permanent. The transition to the final state `s=8` subsumes many different steps of the translation mechanism, which start with GTP hydrolysis and end with elongation of the polypeptide chain with the amino acid carried by the aa-tRNA. Non-cognates never pass beyond state `s=2`, but the outcome of the translation can still be an error if aa-tRNA is near-cognate (*i.e.*, if Boolean `near` is true). In this case, an amino acid is inserted that does not correspond to the codon in the genetic code.

```
// EF-G binding
[ ] (s=6) -> k6f : (s'=7) ;
[ ] (s=7) -> k7b : (s'=6) ;
// GTP hydrolysis, unlocking, tRNA movement and Pi release,
// rearrangements of ribosome and EF-G, dissociation of GDP
[ ] (s=7) -> k7f : (s'=8) ;
```

The model is completed by transitions from the dissociation state `s=0` and the rejection state `s=5` back to the start state `s=1`. After an aa-tRNA is rejected, the race of the four aa-tRNA types resumes. Also, for technical reasons, a self-loop at the final state s=8 is added.

```
// no entrance, re-entrance at state 1
[ ] (s=0) -> FAST : (s'=1) ;
// rejection, re-entrance at state 1
[ ] (s=5) -> FAST : (s'=1) ;
// elongation
[ ] (s=8) -> FAST : (s'=8) ;
```

The rates that are used in our model are given in Table 39.4. They are collected from the biological literature [31, 13].

The complete Prism model can be found in the appendix as Listing A.0.2. In the sequel we use the Prism model described above for the analysis of the probability for insertion errors (*i.e.*, the chance that the peptidyl chain is extended with an amino acid that differs from the one encoded by the codon that is translated).

### 39.4.5 Insertion Errors

Once we have the model, we can use the model checking capabilities of the Prism tool to predict the misreading frequencies for individual codons. To this end we need to give Prism the exact property that corresponds to our question about the probability. In other words, we need the right formula with the above-mentioned CSL. The formula should state that we want to compute the probability that an erroneous state is reached in which a wrong amino acid is added.

For a codon under translation, a pseudo-cognate anticodon carries precisely the amino acid that the codon codes for. Therefore, successful matching of a pseudo-cognate does not lead to an insertion error.

Taking into account the above we come up with the following CSL formula:

```
P=? [ (true) U ((s=8) and not near) ]
```

The formula is of the form P=? [$\Phi$], which is a basic formula template for CSL. The part P=? means that we want a numerical result (*i.e.*, the cumulative probability of all paths that satisfy formula $\Phi$). Like the formulas for the LTL in standard model checking, CSL formulas are also interpreted on sequences of states (*i.e.*, paths) of the model. So, the inner formula $\Phi$ states that we are interested only in paths that end in state s=8—in which the amino acid is added to the chain—and, moreover, that the added amino acid is the wrong one (*i.e.*, the tRNA is not cognate or pseudo-cognate but near-cognate). This last fact is expressed as near, where not is the negation operator. The paths start by default in the initial state s=0. The formula $\Phi$ itself is of the form $\Phi_1$ U $\Phi_2$, where U is the so-called until operator.[6] The meaning

[6] Actually, this operator exists also in LTL, but we "hid" it in the temporal operators [ ] and $\langle\,\rangle$. The latter are just syntactic sugar, and they can be expressed using the until operator $U$.

**Table 39.5 Probabilities per codon for erroneous elongation**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| UUU | 27.4e-4 | CUU | 46.7e-4 | GUU | 1.12e-10 | AUU | 14.4e-4 |
| UUC | 91.2e-4 | CUC | 13.6e-4 | GUC | 55.0e-4 | AUC | 35.0e-4 |
| UUG | 7.59e-4 | CUG | 4.49e-4 | GUG | 2.68e-4 | AUG | 58.3e-4 |
| UUA | 23.5e-4 | CUA | 18.9e-4 | GUA | 22.3e-4 | AUA | 34.4e-4 |
| UCU | 2.81e-10 | CCU | 34.1e-4 | GCU | 1.77e-10 | ACU | 2.73e-10 |
| UCC | 56.1e-4 | CCC | 10.4e-4 | GCC | 12.5e-4 | ACC | 34.2e-4 |
| UCG | 20.3e-4 | CCG | 37.6e-4 | GCG | 3.187e-4 | ACG | 31.7e-4 |
| UCA | 9.09e-4 | CCA | 22.8e-4 | GCA | 28.2e-4 | ACA | 29.1e-4 |
| UGU | 6.97e-4 | CGU | 1.21e-10 | GGU | 1.32e-10 | AGU | 8.70e-4 |
| UGC | 30.4e-4 | CGC | 4.59e-4 | GGC | 9.40e-4 | AGC | 37.2e-4 |
| UGG | 39.8e-4 | CGG | 88.7e-4 | GGG | 2.72e-10 | AGG | 140.7e-4 |
| UGA | 7.50e-4 | CGA | 3.98e-4 | GGA | 100.3e-4 | AGA | 48.1e-4 |
| UAU | 2.81e-10 | CAU | 91.1e-4 | GAU | 18.6e-4 | AAU | 15.2e-4 |
| UAC | 15.7e-4 | CAC | 47.5e-4 | GAC | 43.2e-4 | AAC | 49.3e-4 |
| UAG | 41.3e-4 | CAG | 69.4e-4 | GAG | 7.09e-4 | AAG | 32.1e-4 |
| UAA | 6.04e-4 | CAA | 22.7e-4 | GAA | 21.4e-4 | AAA | 14.6e-4 |

of this kind of formula is that along the path, formula $\Phi_1$ must hold until a state is reached in which formula $\Phi_2$ holds. If $\Phi_2$ holds in the initial state, $\Phi_1$ does not need to hold in that state, because in this case, the formula is trivially true. In our case we have set $\Phi_1$ to `true`. Because `true` holds trivially in all states, this means that we do not care about the intermediate states of the path and that it is only important that a state is reached in which $\Phi_2$ holds (*i.e.*, a wrong amino acid is added to the chain).

Our results obtained with Prism are given in Table 39.5. Prism produces these results within a couple of minutes. Checking for an individual codon takes just a few seconds.

As reported in [13], the probability for an erroneous insertion is strongly correlated with the quotient of the number of near-cognate anticodons and the number of cognate anticodons.This can be seen also in Figure 39.6. On the y-axis is the quotient of the concentrations (number of molecules) of near-cognate and cognate tRNAs, whereas on the x-axis are the probabilities for erroneous insertion.

### 39.4.6 Concluding Remarks

We showed how probabilistic model checking can be used to analyze biological networks as an alternative for Gilliespie-like simulation. As an example we discussed a stochastic model of the translation process based on realistic data of ribosome kinetics. We used the probabilistic model checker Prism and in particular its capabilities to deal with continuous time Markov chains. Compared with simulation, our approach is computationally more reliable as it is independent on the number of simulations. Also, it this case, it has faster response times, taking seconds rather than minutes or hours.

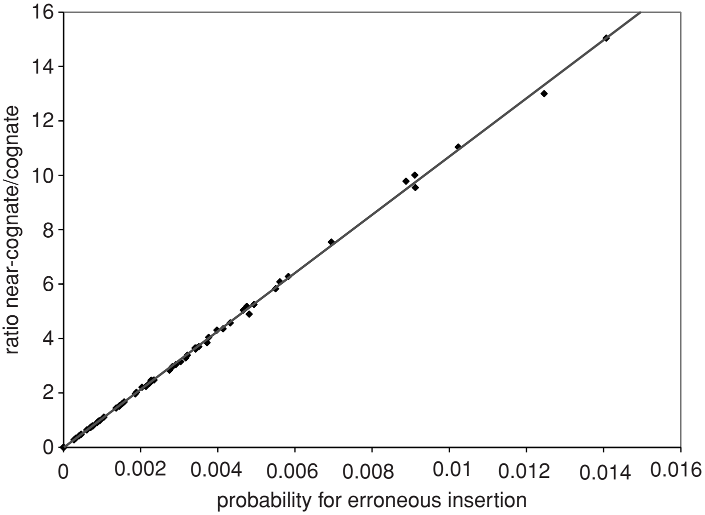


**Figure 39.6** Correlation of ratio near-cognate versus cognates aa-tRNAs and error probabilities.

The kind of probabilistic/stochastic models, as we presented here, has opened new avenues for future work on biological systems that possess intrinsically probabilistic properties. For example current research using the model-checking-based method is targeted at biological processes that require high precision, like DNA translation, DNA repair, charging of the tRNAs with amino acids, and so on. In [4], we show how with our model one could check if amino acids with similar biochemical properties substitute erroneously for one another with greater probabilities than dissimilar ones.

### 39.4.7 Related Work and Bibliographic Notes

The model that is used in this chapter builds on [5], which was inspired by the simulation experiments of mRNA translation reported in [13]. A similar model, based on ordinary differential equations, was developed in [17]. Although probabilistic, it is used to compute insertion times but no translation errors. The model of mRNA translation in [14] assumes insertion rates that are directly proportional to the mRNA concentrations but assigns the same probability of translation error to all codons.

Applications of probabilistic model checking and in particular Prism can be found in [24]. More about probabilistic model checking and the underlying algorithms can be found in [23].

There exist numerous applications of formal methods to biological systems. A selection of recent papers from model checking and process algebra includes [29, 8, 10]. More specifically pertaining to this chapter, [7] applies the Prism model checker to analyze stochastic models of signaling pathways. Their methodology is presented as a more efficient alternative to ordinary differential equations models, including properties that are not of a probabilistic nature. Also, [16] employs Prism on various types of biological pathways, showing how the advanced features of the tool can be exploited to tackle large models.

Prism is a free available software and can be downloaded from its web page `http://www.prismmodelchecker.org`. Of course, any model checking tool that supports CTMCs can be used also for analyzing biological systems. One such tool is MRMC, which is also in the public domain; see `http://www.mrmc-tool.org`.

## REFERENCES

1. M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. Model building and model checking for biochemical processes. *Cell Biochem Biophys*, 38:271–286, 2003.
2. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
3. G. Batt, D. Bergamini, H. de Jong, H. Garavel, and R. Mateescu. Model checking genetic regulatory networks using GNA and CADP. *Proceedings of the 11th SPIN Workshop Barcelone, Spain*, volume 2989 of *Lecture Notes in Computer Science*, 2004.
4. D. Bošnački, H.M.M. ten Eikelder, M.N. Steijaert, and E.P. de Vink. Stochastic analysis of amino acid substitution in protein synthesis. In M. Heiner and A.M. Uhrmacher, editors, *Proceedings of the CMSB 2008*, LNCS 5307, 2008, pp. 367–386.
5. D. Bošnački, T.E. Pronk, and E.P. Vink. In silico modelling and analysis of ribosome kinetics and aa-tRNA competition. *Trans Comput Syst Biol*, IX:69–89, 2009. CompMod 2008 special issue, R.-J. Back and I. Petre, guest editors.
6. M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using the prism model checker. *Proceedings of the CMSB*, 2005, pp. 179–190.
7. M. Calder, V. Vyshemirsky, D. Gilbert, and R. Orton. Analysis of signalling pathways using continuous time Markov chains. *Trans Comput Syst Biol*, VI:44–67, 2006.
8. N. Chabrier and F. Fages. Symbolic model checking of biochemical networks. *Proceedings of the CMSB 2003*, LNCS 2602, 2003, pp. 149–162.
9. C. Chaouiya, E. Remy, P. Ruet, and D. Thieffry. Qualitative modelling of genetic networks: From logical regulatory graphs to standard petri nets. In J. Cortadella and W. Reisig, editors, *ICATPN*, volume 3099 of *Lecture Notes in Computer Science*. Springer, 2004.
10. V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based modelling of cellular signalling. *Proceedings CONCUR*, LNCS 4703, 2007, pp. 17–41.
11. H. Dong, L. Nilsson, and C.G. Kurland. Co-variation of tRNA abundance and codon usage in *Escherichia coli* at different growth rates. *J Mol Biol*, 260:649–663, 1996.
12. E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Formal Models and Semantics*, Elsevier, New York, 1990, pp. 995–1072.
13. A. Fluitt, E. Pienaar, and H. Viljoen. Ribosome kinetics and aa-tRNA competition determine rate and fidelity of peptide synthesis. *Comput Biol Chem*, 31:335–346, 2007.
14. M.A. Gilchrist and A. Wagner. A model of protein translation including codon bias, nonsense errors, and ribosome recycling. *J Theor Biol*, 239:417–434, 2006.
15. K.B. Gromadski and M.V. Rodnina. Kinetic determinants of high-fidelity tRNA discrimination on the ribosome. *Mol Cell*, 13(2):191–200, 2004.

16. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Proceedings of the CMSB 2006*, LNBI 4210, 2006, pp. 32–47.
17. A.W. Heyd and D.A. Drew. A mathematical model for elongation of a peptide chain. *Bull Math Biol*, 65:1095–1109, 2003.
18. G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, Reading, MA, 2003.
19. E.M. Clarke Jr., O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 2000.
20. N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, E.J. Albert Hubbard, and M.J. Stern. Formal modeling of c. elegans development: A scenario-based approach. In C. Priami, editor, *Proceedings of the CMSB*, volume 2602 of *Lecture Notes in Computer Science*, Springer, New York, 2003, pp. 4–20.
21. G. Karp. *Cell and Molecular Biology*, 5th edition, Wiley, New York, 2008.
22. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model cheking with Prism: a hybrid approach. *Int J Softw Tools Technol Transf*, 6:128–142, 2004. See also http://www.prismmodelchecker.org/.
23. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, Springer, New York, 2007, pp. 220–270.
24. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking for systems biology. *Symbolic Systems Biology*. Jones and Bartlett, 2009. To appear.
25. P. Lecca and C. Priami. Cell cycle control in eukaryotes: A biospi model. *Proceedings of the Workshop on Concurrent Models in Molecular Biology (BioConcur'03)*, ENTCS, 2003.
26. H.H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *PNAS*, 94:814–819, 1997.
27. L. Mendoza, D. Thieffry, and E.R. Alvarez-Buylla. Genetic control of flower morphogenesis in arabidopsis thaliana: A logical analysis. *Bioinformatics*, 15(7/8):593–606, 1999.
28. T. Pape, W. Wintermeyer, and M. Rodnina. Complete kinetic mechanism of elongation factor Tu-dependent binding of aa-tRNA to the A-site of *E. coli*. *EMBO J*, 17:7490–7497, 1998.
29. C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to represent ation and simulation of molecular processes. *Inf Proces Lett*, 80:25–31, 2001.
30. A. Regev. *Computational Systems Biology: A Calculus for Biomolecular Knowledge*. PhD thesis, Tel Aviv University, December 2002.
31. M.V. Rodnina and W. Wintermeyer. Ribosome fidelity: tRNA discrimination, proofreading and induced fit. *Trends Biochem Sci*, 26(2):124–130, 2001.
32. A. Savelsbergh, V.I. Katunin, D. Mohr, F. Peske, M.V. Rodnina, and W. Wintermeyer. An elongation factor G-induced ribosome rearrangement precedes tRNA–mRNA translocation. *Mol Cell*, 11:1517–1523, 2003.
33. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Formal Models and Semantics*, Elsevier, New York, 1990, pp. 995–1072.

## APPENDIX

Listing A.0.1 (Promela description of the *Arabidopsis* network)

```
#define T       0
#define L       1
#define A       2
#define G       3
#define P       4
#define I       5

bool Active[6];

proctype TLF1() {
do
:: atomic{Active[T] && !Active[L] -> Active[T]=0}
:: atomic{Active[T] && Active[L] -> Active[T]=0}
od; }

proctype LFY() {
do
:: atomic{Active[L] && !Active[A] && !Active[T] -> Active[L]=0}
:: atomic{Active[L] && !Active[A] && Active[T] -> Active[L]=0}
:: atomic{Active[L] && Active[A] && !Active[T] -> Active[L]=0}
:: atomic{Active[L] && Active[A] && Active[T] -> Active[L]=0}
od; }

proctype AP1() {
do
:: atomic{!Active[A] && !Active[L] && !Active[G] -> Active[A]=1}
:: atomic{!Active[A] && Active[L] && !Active[G] -> Active[A]=1}
:: atomic{Active[A] && !Active[L] && Active[G] -> Active[A]=0}
:: atomic{Active[A] && Active[L] && Active[G] -> Active[A]=0}
od; }

proctype AG() {
do
:: atomic{!Active[G] && !Active[T] && !Active[L] && !Active[A] -> Active[G]=1}
:: atomic{!Active[G] && !Active[T] && Active[L] && !Active[A] -> Active[G]=1}
:: atomic{Active[G] && !Active[T] && !Active[L] && Active[A] -> Active[G]=0}
:: atomic{Active[G] && !Active[T] && Active[L] && Active[A] -> Active[G]=0}
:: atomic{Active[G] && Active[T] && !Active[L] && !Active[A] -> Active[G]=0}
:: atomic{Active[G] && Active[T] && !Active[L] && Active[A] -> Active[G]=0}
:: atomic{Active[G] && Active[T] && Active[L] && !Active[A] -> Active[G]=0}
:: atomic{Active[G] && Active[T] && Active[L] && Active[A] -> Active[G]=0}
od; }

proctype AP3() {
do
:: atomic{!Active[P] && !Active[I] && Active[L] -> Active[P]=1}
:: atomic{!Active[P] && Active[I] && Active[L] -> Active[P]=1}
:: atomic{Active[P] && !Active[I] && !Active[L] -> Active[P]=0}
:: atomic{Active[P] && !Active[I] && Active[L] -> Active[P]=0}
od; }

proctype PI() {
do
:: atomic{!Active[I] && !Active[P] && Active[L] -> Active[I]=1}
:: atomic{!Active[I] && Active[P] && Active[L] -> Active[I]=1}
:: atomic{Active[I] && !Active[P] && !Active[L] -> Active[I]=0}
:: atomic{Active[I] && !Active[P] && Active[L] -> Active[I]=0}
od; }

init {
atomic{
  Active[L]=1;
  Active[A]=1;
  run TLF1();
  run LFY();
  run AP1();
  run AG();
  run AP3();
  run PI();
  }
}
```

Listing A.0.2 (Prism model of mRNA translation)

```
stochastic

// constants
const double ONE=1;
const double FAST=1000;

// tRNA rates
const double c_cogn ;
const double c_pseu ;
const double c_near ;
const double c_nonc ;

const double k1f = 140;
const double k2b =  85;
const double k2bx=2000;
const double k2f = 190;
const double k3bc=   0.23;
const double k3bp=  80;
const double k3bn=  80;
const double k3fc= 260;
const double k3fp=   0.40;
const double k3fn=   0.40;
const double k4rc=  60;
const double k4rp=FAST;
const double k4rn=FAST;
const double k4fc= 166.7;
const double k4fp=  46.1;
const double k4fn=  46.1;
const double k6f = 150;
const double k7b = 140;
const double k7f = 145.8;

module ribosome

s : [0..8] init 1 ;
cogn : bool init false ;
pseu : bool init false ;
near : bool init false ;
nonc : bool init false ;

// initial binding
[ ] (s=1) -> k1f * c_cogn : (s'=2) & (cogn'=true) ;
[ ] (s=1) -> k1f * c_pseu : (s'=2) & (pseu'=true) ;
[ ] (s=1) -> k1f * c_near : (s'=2) & (near'=true) ;

[ ] (s=2) -> k2b : (s'=0) &
      (cogn'=false) & (pseu'=false) & (near'=false) ;

// codon recognition
[ ] (s=2) &      -> k2f : (s'=3) ;
[ ] (s=3) & cogn -> k3bc : (s'=2) ;
[ ] (s=3) & pseu -> k3bp : (s'=2) ;
[ ] (s=3) & near -> k3bn : (s'=2) ;

```

```
// GTPase activation, GTP hydrolysis, reconformation
[ ] (s=3) & cogn -> k3fc : (s'=4) ;
[ ] (s=3) & pseu -> k3fp : (s'=4) ;
[ ] (s=3) & near -> k3fn : (s'=4) ;

// rejection
[ ] (s=4) & cogn -> k4rc : (s'=5) & (cogn'=false) ;
[ ] (s=4) & pseu -> k4rp : (s'=5) & (pseu'=false) ;
[ ] (s=4) & near -> k4rn : (s'=5) & (near'=false) ;


// accommodation, peptidyl transfer
[ ] (s=4) & cogn -> k4fc : (s'=6) ;
[ ] (s=4) & pseu -> k4fp : (s'=6) ;
[ ] (s=4) & near -> k4fn : (s'=6) ;

// EF-G binding
[ ] (s=6) -> k6f : (s'=7) ;
[ ] (s=7) -> k7b : (s'=6) ;

// GTP hydrolysis, unlocking,
// tRNA movement and Pi release,
// rearrangements of ribosome and EF-G,
// dissociation of GDP
[ ] (s=7) -> k7f : (s'=8) ;

// no entrance, re-entrance at state 1
[ ] (s=0) -> FAST : (s'=1) ;
// rejection, re-entrance at state 1
[ ] (s=5) -> FAST : (s'=1) ;
// elongation
[ ] (s=8) -> FAST : (s'=8) ;

endmodule
```

# 40

# REVERSE ENGINEERING OF MOLECULAR NETWORKS FROM A COMMON COMBINATORIAL APPROACH

Bhaskar DasGupta, Paola Vera-Licona, and Eduardo Sontag

## 40.1 INTRODUCTION

The understanding of molecular cell biology requires insight into the structure and dynamics of networks that are made up of thousands of interacting molecules of DNA, RNA, proteins, metabolites, and other components. One of the central goals of systems biology is the unraveling of the as yet poorly characterized complex web of interactions among these components. This work is made harder by the fact that new species and interactions are continuously discovered in experimental work, necessitating the development of adaptive and fast algorithms for network construction and updating. Thus, the "reverse engineering" of networks from data has emerged as one of the central concern of systems biology research.

A variety of reverse-engineering methods have been developed, based on tools from statistics, machine learning, and other mathematical domains. To use these methods effectively, it is essential to develop an understanding of the fundamental characteristics of these algorithms. With that in mind, this chapter is dedicated to the reverse engineering of biological systems.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Specifically, we focus our attention on a particular class of methods for reverse engineering, namely those that rely algorithmically on the so-called "hitting set" problem, which is a classical combinatorial and computer science problem, Each of these methods uses a different algorithm to obtain an exact or an approximate solution of the hitting set problem. We will explore the ultimate impact that the alternative algorithms have on the inference of published *in silico* biological networks.

## 40.2 REVERSE-ENGINEERING OF BIOLOGICAL NETWORKS

Systems biology aims at a systems-level understanding of biology, viewing organisms as integrated and interacting networks of genes, proteins, and other molecular species through biochemical reactions that result in particular form and function (phenotype). Under this "system" conceptualization, it is the interactions among components that give rise to emerging properties.

Systems-level ideas have been a recurrent theme in biology for several decades, as exemplified by Cannon's work on homeostasis [7], Wiener's biological cybernetics [31], and Ludwig von Bertalanffy's foundations of general systems theory [30]. So what has brought systems biology to the mainstream of biological science research in recent years? The answer can be found in large part in enabling technological advances, ranging from high-throughput biotechnology (gene expression arrays, mass spectrometers, *etc.*) to advances in information technology, that have revolutionized the way that biological knowledge is stored, retrieved, and processed.

A systems approach to understanding biology can be described as an iterative process that includes; (i) data collection and integration of all available information (ideally, regarding all the components and their relationships in the organism of interest), (ii) system modeling, (iii) experimentation at a global level, and (iv) generation of new hypotheses (see Figure 40.1).

The current chapter focuses on the system modeling aspects and, specifically, on the top-down modeling approach broadly known as the biological "reverse-engineering," which can be very broadly described as follows:

> *The biological reverse-engineering problem is that of analyzing a given system in order to identify, from biological data, the components of the system and their relationships.*

In broad terms, there are two very different levels of representation for biological networks. They are described as follows.

NETWORK TOPOLOGY REPRESENTATIONS. Also known as "wiring diagrams" or "static graphs," these are coarse diagrams or maps that represent the connections (physical, chemical, or statistical) among the various molecular components of a network. At this level, no detailed kinetic information is included. A network of molecular interactions can be viewed as a graph: Cellular components are nodes in a network, and the interactions (binding, activation, inhibition, *etc.*) between these components are the edges that connect the nodes. A reconstruction of network

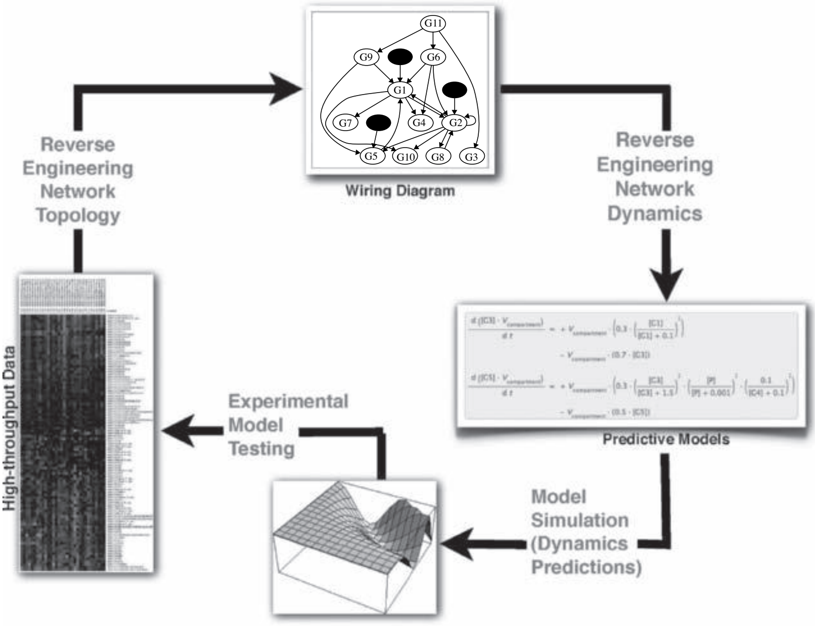

G11
G9
G6
G1
G7
G4
G2
G5
G10
G8
G3
Wiring Diagram
Reverse Engineering Network Topology
Reverse Engineering Network Dynamics
High-throughput Data
Predictive Models
Experimental Model Testing
Model Simulation (Dynamics Predictions)

**Figure 40.1** Iterative process in systems biology.

topology allows one to understand properties that might remain hidden without the model or with a less relevant model.

These type of models can be enriched by adding information on nodes or edges. For instance, "+" or "−" labels on edges may be used in order to indicate positive or negative regulatory influences. The existence of an edge might be specified as being conditional on the object being studied (for instance, a cell) being in a specific global state, or on a particular gene that regulates that particular interaction being expressed above a given threshold. These latter types of additional information, however, refer implicitly to notions of state and temporal evolution, and thus, they lead naturally toward qualitative dynamical models.

Different reverse-engineering methods for topology identification differ on the types of graphs considered. For example, in the work in [3, 9, 11, 24, 26, 27, 32, 33], edges represent statistical correlation between variables. In [10, 13, 15, 17], edges represent causal relationships among nodes.

NETWORK DYNAMICAL MODELS. Dynamical models represent the time-varying behavior of the different molecular components in the network and, thus, provide a more accurate representation of biological function.

Models can be used to simulate the biological system under study. Different choices of values for parameters correspond either to unknown system characteristics or to environmental conditions. The comparison of simulated dynamics with experimental measurements helps refine the model and provide insight on qualitative properties of behavior, such as the identification of steady states or limit cycles, multistable (*e.g.*, switch-like) behavior, the characterization of the role of various parts of the network in terms of signal processing (such as amplifiers, differentiators and integrators, and logic gates), and the assessment of robustness to environmental changes or genetic perturbations.

Examples of this type of inference include those leading to various types of Boolean networks [2, 20–22] or systems of differential equations [12, 16, 28], as well as multistate discrete models [19].

Depending on the type of network analyzed, data availability and quality, network size, and so forth, the different reverse-engineering methods offer different advantages and disadvantages relative to each other. In Section 40.3.1, we will explore some of the common approaches to their systematic evaluation and comparison.

### 40.2.1 Evaluation of the Performance of Reverse-Engineering Methods

The reverse-engineering problem is by its very nature highly “ill-posed,” in the sense that solutions will be far from unique. This lack of uniqueness stems from the many sources of uncertainty: measurement error, lack of knowledge of all the molecular species that are involved in the behavior being analyzed (“hidden variables”), stochasticity of molecular processes, and so forth. In that sense, reverse-engineering methods can at best provide approximate solutions for the network that one wishes to reconstruct, making it very difficult to evaluate their performance through a theoretical study. Instead, their performance is usually assessed empirically, in the following two ways:

**Experimental testing of predictions.** After a model has been inferred, the newly found interactions or predictions can be tested experimentally for network topology and network dynamics inference, respectively.

**Benchmarking testing.** This type of performance evaluation consists on measuring how “close” the method of our interest is from recovering a known network, referred to as the **“gold standard”** for the problem. In the case of dynamical models, one evaluates the ability of the method of interest to reproduce observations that were not taken into account in the “training” phase involved in the construction of the model. On the another hand, for methods that only reconstruct the network topology (wiring diagram), a varierty of standard metrics may be applied.

METRICS FOR NETWORK TOPOLOGY BENCHMARKING. Suppose that $\Gamma$ is the graph representing the network topology of a chosen “gold standard” network. Let $\Gamma_i$ be the graph representing the inferred network topology. Each one of the interactions in $\Gamma_i$ can be classified into one of the these four classes, when comparing with the gold standard:

(a) Correct interactions inferred (true positives, TP)

(b) Incorrect interactions inferred (false positives, FP)

(c) Correct noninteractions inferred (true negatives, TN)

(d) Incorrect noninteractions inferred (false negatives FN)

From this classification of the interactions, we compute the following metrics:

- The Recall or True Positive Rate TPR = TP/(TP + FN).
- The False Positive Rate FPR = FP/(FP + TN).
- The Accuracy ACC = (TP + TN)/Tot$I$, where Tot$I$ is the total number of possible interactions in a network.
- The Precision or Positive Predictive Value PPV = TP/(TP + FP).

As mentioned, the reverse-engineering problem is underconstrained. Every algorithm will have one or more free parameters that helps select a "best" possible prediction. Hence, a more objective evaluation of performance has to somehow involve a range of parameter values. One way to evaluate performance across ranges of parameters is the **receiver operating characteristic (ROC)** method, based on the plot of FPR vs. TPR values. The resulting **ROC plot** depicts relative trade-offs between true positive predictions and false positive prediction across different parameter values (see Figure. 40.2). A closely related approach is the **Recall-Precision plot**, obtained by plotting TPR vs. PPV values.

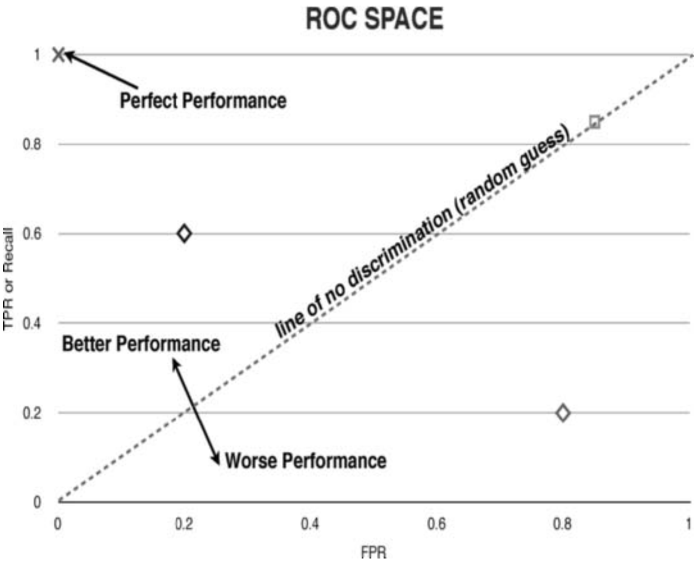


**Figure 40.2** Receiver operating characteristic (ROC)-space. Defined by FPR vs. TPR values in a two-dimensional coordinate system: A perfect reverse-engineering method will ideally have score (FPR, TPR) = (0, 1), whereas the worst possible network will have coordinates (FPR, TPR) = (1, 0), and scores below the identity line (diagonal) indicate methods that perform no better than a random guess.

## 40.3 CLASSICAL COMBINATORIAL ALGORITHMS: A CASE STUDY

We have briefly discussed some basic aspects of reverse engineering of biological systems. Next, as a case of study, we focus our attention on some reverse-engineering algorithms that rely on the solution of the so-called "hitting set problem." The hitting set problem is a classical problem in combinatorics and computer science. It is defined as follows:

**Problem 40.1 (HITTING SET Problem)** *Given a collection $\mathcal{H}$ of subsets of $E = \{1, \ldots, n\}$, find the smallest set $L \subseteq E$ such that $L \cap \mathcal{X} \neq \emptyset$ for all $\mathcal{X} \in \mathcal{H}$.*

The hitting set problem is NP-hard, as can be shown via transformation from its dual, the (minimum) set cover problem [14].

We next introduce some reverse-engineering methods based on the hitting set approach.

- Ideker *et al.* [15].
  This paper introduces two methods to infer the topology of a gene regulatory network from gene expression measurements. The first "network inference" step consists of the estimation of a set of Boolean networks consistent with an observed set of steady-state gene expression profiles, each generated from a different perturbation to the genetic network studied. Next, an "optimization step" involves the use of an entropy-based approach to select an additional perturbation experiment in order to perform a model selection from the set of predicted Boolean networks. In order to compute the sparsest network that interpolates the data, Ideker *et al.* rely on the "minimum set cover" problem. An approximate solution for the hitting set problem is obtained by means of a branch and bound technique [25]. Assessment is performed "*in Numero*": The proposed method is evaluated on simulated networks with varying number of genes and numbers of interactions per gene.
- Jarrah *et al.* [13]
  This paper introduces a method for the inference of the network topology from gene expression data, from which one extracts state transition measurements of wild-type and perturbation data. The goal of this reverse-engineering algorithm is to output one or more most likely network topologies for a collection $x_1, \ldots, x_n$ of molecular species (genes, proteins, *etc.*), which we will refer to as variables. The state of a molecular species can represent its levels of activation. That is, each variable $x_i$ takes values in the set $X = \{0, 1, 2, \ldots\}$ and the interactions among species indicate causal relationships among molecular species. The inference algorithm takes as input one or more time courses of observational data. The output is a most likely network structure for the interactions among $x_1, \ldots, x_n$ that is consistent with the observational data: The notion of consistency with observational data makes the assumption that the regulatory network for $x_1, \ldots, x_n$ can be viewed as a dynamical system

that is described by a function $f : X_n \to X_n$, which transforms an input state $(s_1, \ldots, s_n)$, $s_i \in X$, of the network into an output state $(t_1, \ldots, t_n)$ at the next time step. A directed edge $x_i \to x_j$ in the graph of the network topology of this dynamical system $f$ indicates that the value of $x_j$ under application of $f$ depends on the value of $x_i$. Hence a directed graph is consistent with a given time course $s_1, \ldots, s_r$ of states in $X_n$, if it is the network topology of a function $f : X_n \to X_n$ that reproduces the time course; that is, $f(s_i) = s_{i+1}$ for all $i$.

One possible drawback of reverse-engineering approaches lies in the fact that they construct the "sparest" possible network consistent with the given data. However, real biological networks are known to be not minimal [29]. Although accurate measures of deviation from sparsity are difficult to estimate, nonetheless it seems reasonable to allow additional edges in the network in a "controlled" manner that is consistent with the given data. As already commented in [15], it is possible to add redundancies to the reverse-engineering construction. The basic hitting set approach provides only a minimal set of connections, whereas real biological networks are known to contain redundancies (*e.g.*, see [22]). To account for this, one can modify the hitting set approach to add redundancies systematically by allowing additional parameters to control the extra connections. Theoretically, in terms of the algorithm this corresponds to a standard generalization of the set-cover problem, known as the set-multicover problem, which is well studied in the literature, and for which approximation algorithms are known [4].

The search for the topologies that interpolate the input data involves directly the hitting set problem, which is solved analytically with the use of a computational algebra tools.

The algorithms presented in [5, 17] also make use of hitting set algorithms, but we will restrict our attention to the comparison of the two methods described above.

### 40.3.1 Benchmarking RE Combinatorial-Based Methods

***40.3.1.1 In Silico Gene Regulatory Networks.*** We use data from two different regulatory networks. These contain some features that are common in real regulatory networks, such as time delays and the need for a measurement data presented into discrete states (0, 1, 2, ...).

IN SILICO NETWORK 40.1: GENE REGULATORY NETWORK WITH EXTERNAL PERTURBATIONS. This network was originally introduced in [6]. It was generated using software package given in [23], the interactions between genes in this regulatory network are phenomenological, and represent the net effect of transcription, translation, and post-translation modifications on the regulation of the genes in the network. The model is implemented as a system of ODEs in *Copasi* [18].

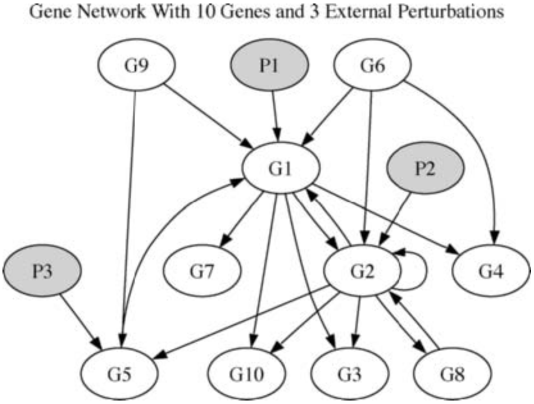


**Figure 40.3** Network 1: 10 genes and 3 environmental perturbations. In this network, the 3 environmental perturbations P1, P2, and P3 directly affect the expression rate of genes G1, G2, and G5, respectively.

This network, shown in Figure 40.3, consists of 13 species: ten genes plus three different environmental perturbations. The perturbations affect the transcription rate of the gene on which they act directly (through inhibition or activation) and their effect is propagated throughout the network by the interactions between the genes.

Network 40.2: segment polarity genes network in *drosophila melanogaster*. The network of segment polarity genes is responsible for pattern formation in the *D. melanogaster* embryo (Figure 40.4). Albert and Othmer [1] proposed and analyzed a Boolean model based on the binary ON/OFF representation of mRNA and protein levels of five segment polarity genes. This model was constructed based on the known topology, and it was validated using published gene and expression data. We generated time courses from this model, from which we will attempt to reverse-engineer the network in order to benchmark the performance of the reverse-engineering algorithms being evaluated.

The network of the segment polarity genes represents the last step in the hierarchical cascade of gene families initiating the segmented body of the fruit fly. The genes of this network include engrailed ($en$), wingless ($wg$), hedgehog ($hh$), patched ($ptc$), cubitus interruptus ($ci$), and sloppy paired ($slp$), coding for the corresponding proteins, which are represented by capital letters (*EN, WG, HH, PTC, CI*, and *SLP*). Two additional proteins, resulting from transformations of the protein $CI$, also play important roles: *CI* may be converted into a transcriptional activator, *CIA*, or may be cleaved to form a transcriptional repressor *CIR*. The expression of the segment polarity genes occurs in stripes that encircle the embryo. These key features of these patterns can be represented in one dimension by a line of 12 interconnected cells, grouped into three parasegment primordia, in which the genes are expressed every fourth cell. In Albert and Othmer [1], parasegments are assumed to be identical, and thus only one parasegment of four cells is considered. Therefore, in the model, the

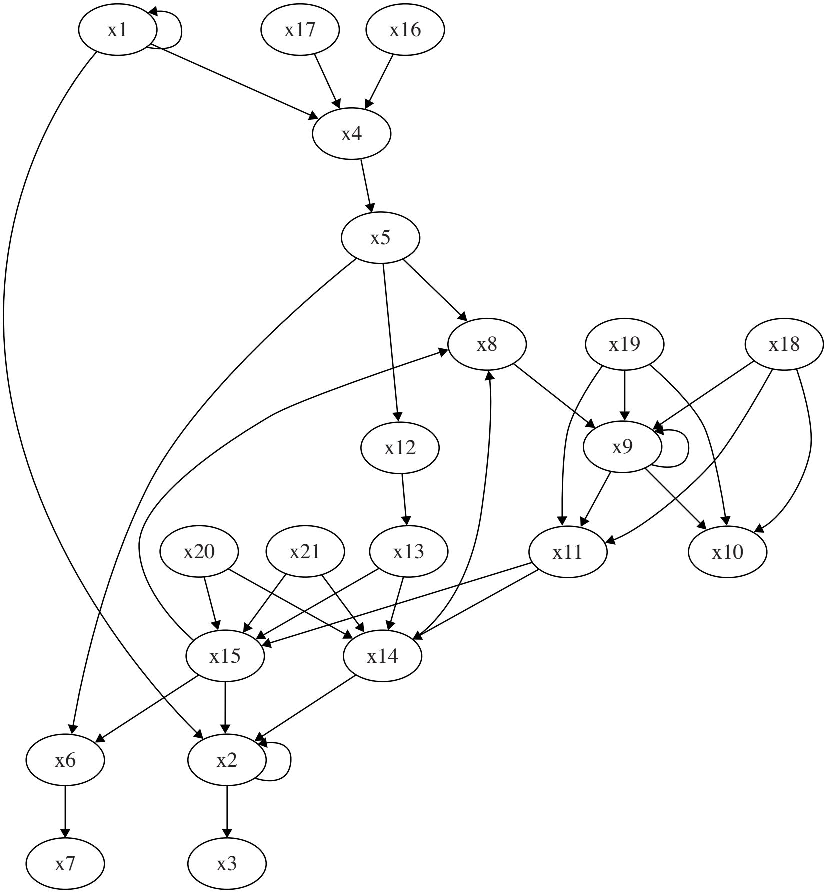


**Figure 40.4** Segment polarity genes network on the *D. melanogaster*. This network consists of the interaction of 60 molecular species: genes and proteins.

variables are the expression levels of the segment polarity genes and proteins (listed above) in each of the four cells, and the network can be seen as a $15 \times 4 = 60$ node network. Using the wild-type pattern from [1], we consider one wild-type time series of length 23.

**40.3.1.2 Results of Comparison.** In this section we compare the results obtained after running Jarrah *et al.*'s and Ideker *et al.*'s methods on each of the above networks. Computations were made on Mac OS X, Processor 2GHz Intel Core 2 Duo.

As we mentioned in Section 40.3, for Jarrah *et al.*'s method, the input data must be discrete. Hence, in order to apply this reverse-engineering method to network 1, we discretize the input data, considering then different discretizations as our running parameter to test Jarrah *et al.*'s method in the ROC space. We specifically use

**Table 40.1 Comparison of RE methods**

| | TP | FP | TN | FN | TPR | FPR | ACC | PPV |
|---|---|---|---|---|---|---|---|---|
| Network1 | | | | | | | | |
| Jarrah Exact Sol D7 | 12 | 34 | 113 | 10 | .5454 | .231 | .7396 | .2608 |
| Jarrah Exact Sol Q5 | 9 | 49 | 98 | 13 | .4090 | .3334 | .6331 | .1551 |
| Jarrah Exat Sol I5 | 7 | 46 | 101 | 15 | .3181 | .313 | .6390 | .1320 |
| Karp Greedy Approx R1 | 9 | 49 | 98 | 13 | .4090 | .666 | .016 | .084 |
| Karp Greedy Approx R2 | 11 | 63 | 84 | 11 | .5 | .571 | .020 | .115 |
| Karp LP Approx. R1 | 7 | 46 | 101 | 15 | .318 | .687 | .014 | .064 |
| Karp LP Approx. R2 | 9 | 59 | 88 | 13 | .409 | .598 | .018 | .092 |
| Network 2 | | | | | | | | |
| Jarrah Exact Sol | – | – | – | – | – | – | – | – |
| Karp Greedy Approx R1 | 4 | 3321 | 91 | 124 | .031 | .026 | .923 | .042 |
| Karp Greedy Approx R2 | 15 | 3254 | 218 | 113 | .117 | .062 | .908 | .064 |
| Karp LP Approx. R1 | 3 | 3279 | 93 | 125 | .023 | .026 | .939 | .031 |
| Karp LP Approx. R2 | 9 | 3285 | 187 | 119 | .070 | 054 | .915 | .045 |

three discretization methods: a graph-theoretic based approached “D” (see [8]), as well as quantile “Q” (discretization method on which each variable state receives an equal number of data values) and interval “I” discretization (discretization method on which we select thresholds for the different discrete values).

For Ideker *et al.*’s method we have considered both greedy and linear programming approximations to the hitting set problem as well as redundancy values (how many extra edges one allows) of $R = 1$ or 2.

We have displayed some our results on Table 40.1. We observe that for network 1, Jarrah *et al.*’s method obtains better results than Ideker *et al.*’s method when considering these values in the ROC space, although both fare very poorly. On the other hand, we observe that Ideker *et al.*’s method achieves a performance no better than random guessing on this network. In contrast, for network 2, Jarrah *et al.*’s method could not obtain any results after running their method for more than 12 hours, but Ideker *et al.*’s method was able to compute results for such a network in less than 1 minute. Also Ideker *et al.*’s method improved slightly its results when the redundancy number is increased; this might indicate the shortcoming of inferring sparser networks when they are of a larger size containing redundancies.

### 40.3.2 Software Availability

The implementation of Jarrah *et al.*’s algorithm [13] is available online through the web interface provided at http://polymath.vbi.vt.edu/polynome/. The implementation of Ideker *et al.*’s algorithm [15] is available online through the web interface provided at http://sts.bioengr.uic.edu/causal/.

## 40.4 CONCLUDING REMARKS

In this chapter, we first provided a brief discussion of the biological reverse-engineering problem, which is a central problem in systems biology. As a case study, we then focused on two methods that rely on the solution of the "hitting set problem" but that differ in their approach to solve this problem, thus leading to different performance.

In terms of network inference power, we hypothesize that, for the smaller network, the poor quality of the results when using Jarrah's approach might be ascribed to the type of data used: In [13], it is claimed that the method performs better if perturbation data are added. The algorithm has the ability of considering both wild-type and mutant data to infer the network, and probably results would improve if using such additional data. In the case of Ideker *et al.*'s method, in both networks we think that it is possible that the low quality of results could be due to the lack of ability of using more than one time series at a time, as well as the fact that the implementation of the method does not include self-loops (self-loops are edges connecting a node to itself that may, for example, represent degradation terms in biochemical systems). We believe that this feature is fundamental for a good performance of the algorithm.

When comparing the computational efficiency of the approaches, one should keep in mind that there will always be a difference between exact solutions and approximate solutions based on greedy algorithms or linear programming relaxations. However, since the size of the networks was fairly small, it is possible that the reason Jarrah's method did not find a solution within a reasonable time might lie in encoding issues rather than in the intrinsic computational complexity of the problem.

## ACKNOWLEDGMENTS

The authors would like to thank Joe Dundas for the implementation and maintenance of the web tool for the Ideker *et al.* method. We would like to thank as well Dr. Brandilyn Stigler for useful discussions on different aspects of this book chapter. This work was supported in part by grants AFOSR FA9550-08, NIH 1R01GM086881, and NSF grants DMS-0614371, DBI-0543365, IIS-0612044, IIS-0346973, and the DIMACS special focus on Computational and Mathematical Epidemiology.

## REFERENCES

1. R. Albert and H. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster. *J Theor Biol*, 223:1–18, 2003.
2. T. Akutsu, S. Miyano, and S. Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, 2000.
3. M.J. Beal and F. Falciani. A Bayesian approach to reconstructing genetic regulatory networks with hidden factors. *Bioinformatics*, 21(3):349–356, 2005.

4. P. Berman, B. DasGupta, and E. Sontag. Randomized approximation algorithms for set multicover problems with applications to reverse-engineering of protein and gene networks. *Discrete Appl Math*, 155(6–7):733–749, 2007.
5. P. Berman, B. DasGupta, and E. Sontag. Algorithmic issues in reverse-engineering of protein and gene networks via the modular response analysis method. *Ann NY Acad Sci*, 1115:132–141, 2007.
6. D. Camacho, P. Vera-Licona, P. Mendes, and R. Laubenbacher. Comparison of reverse-engineering methods using an in silico network. *Proc NY Acad Sci*, 1115(1):73–89, 2007.
7. W.B. Cannon. The wisdom of the body. Norton, New York, 1993.
8. E. Dimitrova, L. Garcia-Puente, A.S. Jarrah, R. Laubenbacher, B. Stigler, M. Stillman, and P. Vera-Licona. Parameter estimation for Boolean models of biological networks. Submitted for publication.
9. N. Dojer, A. Gambin, A. Mizera, B. Wilczynski, and J. Tiuryn. Applying dynamic Bayesian networks to perturbed gene expression data. *BMC Bioinformatics*, 7(1):249, 2006.
10. N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. *J Comput Biol*, 7(3–4):601–620, 2000.
11. A. de la Fuente, N. Bing, I. Hoeschele, and P. Mendes. Discovery of meaningful associations in genomic data using partial correlation coefficients. *Bioinformatics*, 20:3565–3574, 2004.
12. T.S. Gardner, D. di Bernardo, D. Lorez, and J.J. Collins. Inferring genetic networks and identifying compound mode of action via expression profiling. *Science*, 301(5629):102–105, 2003.
13. A.S. Jarrah, R. Laubenbacher, B. Stigler, and M. Stillman. Reverse-engineering polynomial dynamical systems. *Adv Appl Math*, 39(4):477–489, 2007.
14. R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Plenum Press, New York, 1972.
15. T.E. Ideker, V. Thorsson, and R.M. Karp. Discovery of regulatory interactions through perturbation: Inference and experimental design. *Pacific Symposium on Biocomputing*, 2000, pp. 305–316.
16. J. Kim, D. Bates, I. Postlethwaite, P. Heslop-Harrison, and K.H. Cho. Least-squares methods for identifying biochemical regulatory networks from noisy measurements. *BMC Bioinformatics*, 8(1):8, 2007.
17. B. Krupa. On the number of experiments required to find the causal structure of complex systems. *J Theor Biol*, 219(2):257–267, 2002.
18. S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI—a COmplex PAthway SImulator. *Bioinformatics*, 22:3067–3074, 2006.
19. R. Laubenbacher and B. Stigler. A computational algebra approach to the reverse-engineering of gene regulatory networks. *J Theor Biol*, 229:523–537, 2004.
20. S. Liang, S. Fuhrman, and R. Somogyi. Reveal, a general reverse-engineering algorithm for inference of genetic network architectures. *Pacific Symposium on Biocomputing*, 1998, pp. 18–29.
21. S. Martin, Z. Zhang, A. Martino, and J.L. Faulon. Boolean dynamics of genetic regulatory networks inferred from microarray time series data. *Bioinformatics*, 23(7): 866–874, 2007.

22. S. Mehra, W.S. Hu, and G. Karypisb. A Boolean algorithm for reconstructing the structure of regulatory networks. *Metabolic Eng*, 6(4):326, 2004.
23. P. Mendes. Biochemistry by numbers: Simulation of biochemical pathways with Gepasi 3. *Trends Biochem Sci*, 22:361–363, 1997.
24. N. Nariai, Y. Tamada, S. Imoto, and S. Miyano. Estimating gene regulatory networks and protein-protein interactions of Saccharomyces cerevisiae from multiple genome-wide data. *Bioinformatics*, 21(suppl 2):ii206–ii212, 2005.
25. G.L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.
26. I. Pournara and L. Wernisch. Reconstruction of gene networks using Bayesian learning and manipulation experiments. *Bioinformatics*, 20(17):2934–2942, 2004.
27. J.J. Rice, Y. Tu, and G. Stolovitzky. Reconstructing biological networks using conditional correlation analysis. *Bioinformatics*, 21(6):765–773, 2005.
28. E. Sontag, A. Kiyatkin, and B.N. Kholodenko. Network reconstruction based on steady-state data. *Essays Biochem*, 45:161–176, 2008.
29. G. Tononi, O. Sporns, and G.H. Edelman. Measures of degeneracy and redundancy in biological networks. *PNAS*, 96(6):3257–3262, 1999.
30. L. von Bertalanffy. *General System Theory*. Braziler, New York, 1968.
31. N. Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. The MIT Press, Cambridge, MA, 1948.
32. J. Yu, V. Smith, P. Wang, A. Hartemink, and E. Jarvis. Advances to Bayesian network inference for generating causal networks from observational biological data. *Bioinformatics*, 20:3594–3603, 2004.
33. M. Zou and S.D. Conzen. A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, 21(1):71–79, 2005.

# 41

# UNSUPERVISED LEARNING FOR GENE REGULATION NETWORK INFERENCE FROM EXPRESSION DATA: A REVIEW

Mohamed Elati and Céline Rouveirol

## 41.1 INTRODUCTION

In the organism, each organ, each cell, and each protein has a defined role to fulfill so that life is maintained. When one player deviates from its predefined scenario, the consequences are generally not even noticeable. However, sometimes, the consequences are very important, and the whole organism might not survive it. The only way to fight against these disorders is to understand fully the functioning of the whole ensemble, and how each deviation affects it. This is what systems biology is about [2, 14]. At the cellular level, this requires understanding how the different components interact, and how these interactions result in functional networks, often called pathways. Regulation of gene expression at the transcriptional level is a fundamental mechanism that is evolutionarily conserved in all the cellular systems. This form of regulation is typically mediated by transcription factors (TFs) that bind to short DNA sequence motifs, also called binding sites, in promoter regions of transcriptional units and either activate or repress the expression of nearby genes.

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

In the past decade, microarrays and other high-throughput methods have revolutionized biology, providing a wealth of new experimental data. This unprecedented amount of data—for example, the simultaneous measurement of the expression levels of all genes in a given organism—requires the development of new techniques in order to extract biological meaning. One very important aim of such new data analysis approaches is to infer genetic regulatory networks [61] from microarray data (also known as transcriptome data). Genetic regulatory networks, denoted GRNs in the following, are large graphical structures, such that nodes are genes and links are regulation relations. Their inference is a central problem in bioinformatics, but this problem is highly complex, as the number of candidate networks is exponential in the number of genes potentially involved in the network. Machine learning and, in particular, unsupervised machine learning is essential to efficient and tractable network inference (*i.e.*, inference of the network structure and parameters that cause the observed gene expressions and phenotypic states).

There have been numerous efforts in this direction, and this chapter is a review of the main trends in this very active research area. Considerable efforts are being made to chart large-scale gene regulatory networks by relating the expression of a target gene to that of the genes encoding its regulators [67, 25, 43]. Many recent studies have aimed to derive complete genetic regulatory networks in yeast (*Saccharomyces cerevisiae*) using additional information, such as protein-DNA binding from ChIP-chip experiments [45] or computational analysis of transcription factor binding sites [49], with the computational advantage of restricting the number of possible regulators for a given target gene. However, these approaches are difficult to adapt to other organisms, for which the computational detection of binding sites is far less tractable, and the experimental detection of binding events is currently very limited (*e.g.*, *Homo sapiens*). In contrast, the collection of expression datasets is growing at an exponential rate, and methods that rely solely on gene expression for network reconstruction are needed.

The chapter starts with a brief survey about the underlying biology and the data available for GRN inference. Section 41.2 summarizes the biology that underlies the statistical and machine learning problems in the field of network inference. Then, Sections 41.3 and 41.4 discuss available data and introduce the network inference problem. Sections 41.5, 41.6, and 41.7 detail existing approaches for network inference along three axes, from the most simple model for GRN (*i.e.*, co-expression network) to the more sophisticated ones (Bayesian networks and labeled DAGs). Section 41.8 describes several ways to validate the so infered networks. Finally, the chapter is concluded with an analysis of the field as a whole, some underlying methodological issues, and a few possible areas for future research.

## 41.2 GENE NETWORKS: DEFINITION AND PROPERTIES

Gene regulation is a general name for several sequential processes, the most well known and understood being transcription and translation, which control the level of a gene's expression and, ultimately, result in a specific quantity of a target protein. A gene regulation system [4, 42] consists of genes, binding sites, and regulators. The regulators are most often proteins, called transcription factors (denoted as TFs in the following), but small molecules, like RNAs and metabolites, sometimes also participate in the overall regulation. The interactions and binding of regulators to DNA sequence motifs in promoter regions of genes controls the level of gene expression during transcription. The promoter regions serve to aggregate the input signals, mediated by the regulators, and thereby effect a very specific gene expression signal. The traditional roles of individuals or groups of TFs as activators and inhibitors, with respect to a gene, are assigned given the change they cause in expression, and are viable if their effects are strong enough and independent of other TF effects [32]. Those roles have been refined recently by associating function to modules of binding sites (or equivalently TFs) to include more operators, like enhancers, switches, and amplifiers, which form the repertoire of transcriptional control elements. Some of those have been used to engineer synthetic gene circuits with prespecified functions.

In many cases, the level of expression of the gene encoding a transcription factor provides a good indication of its activity. References [55, 59] expanded the set of candidate regulators to proteins involved in various aspects of gene regulation, including both known and putative transcription factors but also signal transduction molecules, to obtain additional information about regulation by considering the levels of expression of signalling molecules with potential indirect effects on transcription. The genes, regulators, and the regulatory connections between them, together with an interpretation scheme, form gene networks (see Figure 41.1).

Substantial knowledge has been gained recently about gene regulation and networks. Some of this knowledge can be used to effectively model gene networks. Gene regulation networks are sparse [39] (*i.e.*, there is a small number of edges per node, much smaller than the total number of nodes). The sparseness property is often used to prune the search space during network inference, as described later. Recent studies [4] have also shown that the frequency distribution of connectivity of nodes

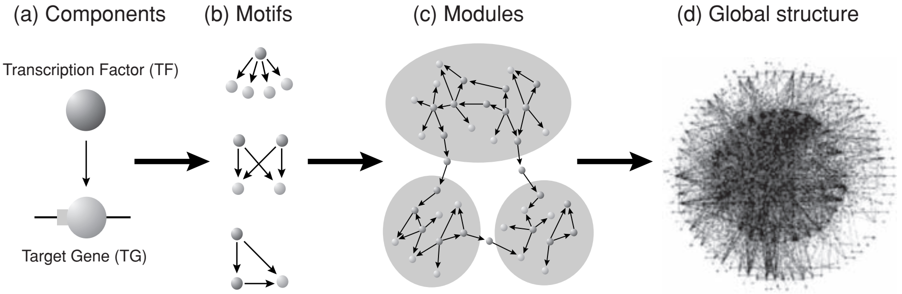


**Figure 41.1** Organization of the transcriptional regulatory network: (a) The basic unit consists of a regulatory interaction between a TF and a target gene. (b) The basic unit forms small patterns of regulatory interactions called network motifs. (c) Module, a set of connected genes that together are involved in a biological process. (d) The set of all transcriptional regulatory interactions is referred to as the GRN.

in biological networks tends to be longer tailed than the normal distribution. The appropriate distribution seems to belong to a class of power-law functions described by $P(k) = k^{-\beta}$, where $k$ is the degree of a vertex in the network graph and $\beta$ is some network specific constant. Such networks are called *scale-free* and exhibit several important properties. The first is the emergence of *hubs*, or highly connected nodes, which are highly unlikely to appear in a network with normally distributed node degrees. The second property is that through the hubs, the rest of the nodes are connected by, in the worst case, very short paths, yielding overall short longest paths between nodes (degree of separation, small-world networks).

## 41.3 GENE EXPRESSION: DATA AND ANALYSIS

The amount of mRNA produced during transcription is a measure of how active or functional a gene is. Gene chips, or microarrays [58], are large-scale gene expression monitoring technologies used to detect differences in mRNA levels of thousands of genes at a time, thus speeding up dramatically genome-level functional studies. Microarrays are used to identify the differential expression of genes between two experiments, typically test versus control, and to identify similarly expressed genes over multiple experiments. An important issue that is often underestimated is the preprocessing of the gene expression data prior to the inference of networks [30]. Although microarray technology produces continuous data, many methods require data discretization prior to further analysis. Data discretization [29, 56] implicitly assumes that in a large compendium of microarrays, the complete dynamic range of expression values was observed for each gene. This complete range can then for example be subdivided into discrete levels such as high, basal, or low expression level [35]. This discretization step is critical due to the potential loss of information. Similarly, interpreting discretization levels as over-, normal, and under-expression should be treated with caution because observing the complete dynamic range of a gene can never be guaranteed unless a large compendium of data is used [41].

The processing pipeline of microarray data involves preprocessing the raw data to get a gene expression matrix and then analyzing the matrix for differences and/or similarities of expression. We will assume in this chapter that the gene expression matrix contains preprocessed expression values with genes as rows and experiments as columns. Thus, each column corresponds to an array or gene-chip. The experiments can be time points (for time-course experiments) or treatments (for perturbation experiments). Each row in the matrix represents a gene expression profile. Gene-chips can hold probes for tens of thousands of genes, whereas the number of experiments, limited by resources, is much smaller, at most in the hundreds. Thus, the gene expression matrix is typically very narrow. This is known as the dimensionality curse, and it is a serious problem for gene network inference. The inference methods described in the following sections use different ways to address this problem.

## 41.4 NETWORK INFERENCE AS AN UNSUPERVISED LEARNING PROBLEM

Machine learning [50] is a scientific discipline concerned with the design and development of algorithms that allow computers to improve their performance based on observation data. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data. Machine learning has proposed a very large range of algorithms. One very important class of such methods is unsupervised learning. Intuitively, unsupervised learning techniques seek to determine how observed data are organized by uncovering regularities/similarities within the data. It is distinguished from supervised learning (and reinforcement learning) in that the learner is given only unlabeled examples. Unsupervised learning is closely related to the problem of density estimation in statistics. However, unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data.

The inference of regulatory networks from data has been addressed by several unsupervised machine learning methods. Three main research directions have been explored in the recent literature:

- Learning regulation networks based on co-regulation detection [15, 28].
- Learning static models of genetic regulation ranging from learning classic Bayesian networks [35] to extensions [55, 59]
- (Constrained) itemset mining [40, 52, 9, 11, 31]

The first type of approach uses relatedness metrics, such as correlation metrics or mutual information to elucidate network structure independently of a gene regulation model. The last two families of network inference methods address the difficult problem of identifying the structure of the regulation network with more complex assumptions. When expressed as a combinatorial problem (*i.e.*, given a set of genes, find the network linking those genes that optimizes some score function), learning a GRN structure is known to be NP-hard [21]. NP-hardness calls for a relaxation of the combinatorial problem or demands heuristics to explore the finite huge set of candidate networks for a given number of genes. Feasibility depends on the training dataset size and the extent of constraints specified by the user through prior knowledge. We describe in the next sections some recent publications in each of these families and how they solve the combinatorial problem of exploring a huge number of candidate graphs.

## 41.5 CORRELATION-BASED METHODS

Co-expression networks (also known as relevance or correlation networks) are graphs where edges connect highly co-expressed genes [15, 28, 26, 5]. Two genes are co-expressed if their expression profiles (rows in the expression matrix) are strongly correlated. They can also be weighted, with the weights indicating the strengths of the relationships. Either the nodes, edges, or both are sometimes labeled with the function, or nature, of the relationship (*i.e.*, activation, inhibition, *etc.*).

The correlation can be measured in different ways. Two of the most important measures are the connected correlation coefficient (or its normalized version, the Pearson correlation coefficient) [18] and the mutual pairwise information of the two variables [15]. Mutual information is more general than the Pearson correlation coefficient. This quantifies only linear dependencies between variables, and a vanishing Pearson correlation does not imply that two variables are statistically independent. In practical application, however, mutual information and Pearson correlation may yield almost identical results [64].

Typically, parsimonious arguments stemming from biological principles are used to restrict the resulting networks. Such networks normally contain much more links than the actual regulatory interaction between transcription factors and regulated genes. Also, second neighbours may still be considerably correlated. Inference algorithms based on this measure are numerous. Some approaches are as follows.

***Relevance Network.*** The relevance network approach [15, 28] has been introduced in gene clustering and was successfully applied to infer relationships between RNA expressions and chemotherapeutic susceptibility. The approach consists of inferring a genetic network where a pair of genes $(g_i, g_j)$ is linked by an edge if the mutual information $I(g_i, g_j)$ is larger than a given threshold. The complexity of the method is $O(n^2)$ because all pairwise interactions are considered. Mutual information, $I(g_i, g_j)$, is computed as

$$\sum P(g_i, g_j) \log \frac{P(g_i, g_j)}{P(g_i)P(g_j)}$$

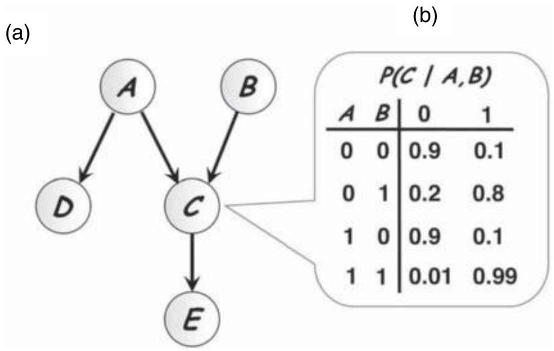
Note that this method does not eliminate all the indirect interactions between genes. For example, if gene $g_1$ regulates both gene $g_2$ and gene $g_3$, this would cause high mutual information among the pairs $(g_1, g_2)$, $(g_1, g_3)$, and $(g_2, g_3)$. As a consequence, the algorithm would set an edge between $g_2$ and $g_3$, although these two genes interact only through gene $g_1$.

***CLR Algorithm.*** The CLR algorithm [33] is an extension of the relevance network approach. This algorithm computes the mutual information for each pair of genes and derives a score related to the empirical distribution of the mutual information values. In particular, instead of considering the information $I(g_i, g_j)$ between genes $g_i$ and $g_j$, it takes into account the sample mean and standard deviation of the empirical distribution of the values $I(g_i, g_k)$, $k = 1, \dots, n$. The CLR algorithm was successfully applied to decipher the *Escherichia coli*. CLR has a complexity in $O(n^2)$ once the mutual information is computed.

***ARACNE Algorithm.*** Reference [5] starts by assigning to each pair of nodes a weight equal to the mutual information. Then, as in relevance networks, all edges for which $I(g_i, g_j) < I_0$ are removed, with $I_0$ a given threshold. Eventually, the weakest edge of each triplet is interpreted as an indirect interaction and is removed if the difference between the two lowest weights is above a threshold $W_0$. Note that by increasing $I_0$ the number of inferred edges is decreased, whereas the opposite effect is obtained by increasing $W_0$. Once $I(g_i, g_j)$ for all gene pairs has been computed, ARACNE excludes all the pairs for which the null hypothesis of mutually independent genes cannot be ruled out. A $p$-value for the null hypothesis, computed using Monte Carlo simulations, is associated with each value of the mutual information. The final step of this algorithm is a pruning step that tries to reduce the number of false-positives. They use the data processing inequality (DPI) principle that asserts that if both $(g_i, g_j)$ and $(g_j, g_k)$ are directly interacting, and $(g_i, g_k)$ is indirectly interacting through $j$, then $I(g_i, g_k) \leq \min(I(g_i, g_j), I(g_j, g_k))$. This elimination step increases strongly the precision of the algorithm since it reduces the number of false-positives, but it decreases also its sensitivity canceling many weak pair interactions. ARACNE's complexity is $O(n^3)$ because the algorithm considers all triplets of genes. In [5], the method was able to recover components of the GRN in mammalian cells and outperformed Bayesian networks and relevance networks on several inference tasks.

## 41.6 PROBABILISTIC GRAPHICAL MODELS

When modeling a gene regulatory network, the goal is to uncover the relationships linking entities that are involved in the system under study (*e.g.*, genes) and their different attributes (*e.g.*, expression level). In a probabilistic model [54], these attributes are handled as random variables. Random variables include observed attributes, such as the expression level of a particular gene in a particular experiment, as well as hidden attributes that are assumed by the model, such as the cluster assignment of a particular gene. A model embodies the description of the joint probability distribution of all the random variables of interest. Probabilistic graphical models represent multivariate joint probability distributions via a product of terms, each of which involves only a few variables. The structure of the product is represented by a graph that relates variables that appear in a common term. This graph specifies the product form of the distribution and provides tools for reasoning about the properties entailed by the product. For a sparse graph, the representation is compact and in many cases allows effective inference. In Bayesian networks, the joint distribution over a set $G = g_1, \ldots, g_n$ of random variables is represented as a product of conditional probabilities. A Bayesian network associates with each variable $g_i$ a conditional probability $P(g_i|P_i)$, where $P_i \subseteq G$ is the set of variables that are called the parents of $g_i$. The resulting product is of the form $P(g_1, \ldots, g_n) = \prod P(g_i|P_i)$. A Bayesian network implicitly encodes the Markov assumption that given its parents, each variable is independent of its nondescendants. The graphical representation is given by a directed acyclic graph where we put edges from $g_i$'s parents to $g_i$. To specify a



P(C | A,B)

| A | B | 0 | 1 |
|---|---|---|---|
| 0 | 0 | 0.9 | 0.1 |
| 0 | 1 | 0.2 | 0.8 |
| 1 | 0 | 0.9 | 0.1 |
| 1 | 1 | 0.01 | 0.99 |

**Figure 41.2** An example of Bayesian network, where the genes can take discrete states of Up (1) and Down (0). The probability distributions at the nodes imply the dependencies of nodes only on their parent nodes. The joint probability can be computed directly from the graph, for any instance of the node values.

model completely, we need to describe the conditional probability associated with each variable (see Figure 41.2). In general, any statistical regression model may be used. For example, models where each $P(g_i|P_i)$ is a linear regression of $g_i$ on $P_i$ can be considered. Alternatively, decision trees can be uses to represent the probability of a discrete variable $g_i$ given the values of its parents [66]. The choice of a specific parametric representation of the conditional probabilities is often dictated by our knowledge or assumptions about the domain.

***Bayesian Networks.*** The general aim is to learn a model that is as close as possible to the underlying distribution from which the observations were made [35, 62]. Two main tasks can be distinguished: parameter estimation and model selection. In parameter estimation, the parameters of the conditional probabilities are learned for a given model structure. This task is often addressed as a maximum likelihood problem [47]. In model selection, the goal is to select among different model structures to find one that best reflects the dependencies in the domain [34, 21]. This task is often addressed as a discrete optimization problem where one tries to maximize a score that measures how well each candidate structure matches the observed data. Once a model that describes a joint distribution has been specified or learned, it can be uses to compute the likelihood of observations and make predictions about the value of unobserved random variables given these observations. There is a wide choice of exact and approximate methods for answering such queries. These exploit, when possible, the structure of the product form for efficient computation. In [35, 62], the authors use Bayesian networks to establish regulatory relationships between genes in yeast, based on time-series data of gene expression. To do that, they used several biologically plausible simplifying assumptions. The first one is that the nodes are of bounded in-degree (*i.e.*, the networks are sparse). The second is that the parent and children nodes likely have similar expression patterns, and thus co-expressed pairs of genes might be in regulatory relationships. Both are realistic to a degree and are meant to reduce the number of potential high-scoring networks. It was found that even with these assumptions, the number of high-scoring networks was still high. The problem with learning of Bayesian networks is combinatorial: If the graph model is not known, then the space of all graph models has to be explored. But this space is super-exponential even for directed acyclic graphs and exploring it completely is impossible even with the fastest heuristics [54, 21]. Exploring hierarchical properties of DAGs can help in pruning the search space from super-exponential down to exponential size for which, with current technology, exact solutions can be obtained for small networks of $n$ ¡ 30 nodes. A promising direction is to restrict the search space further by including additional biological knowledge as it becomes available.

Minreg ***Algorithm.*** The authors of [55] have designed the Minreg system, a constrained Bayesian network for the reconstruction of regulatory networks. The maximal in-degree of target genes and the total number of regulators in the model are limited, so the model focuses on only a small set of global active regulators (ARs). The authors have made use of these constraints to devise an approximation algorithm to search for a high scoring network given expression data. The system successfully and robustly identifies the key active regulators but cannot learn the full detailed network and may miss interesting regulation relationships: Given a current set of ARs, the greedy search of Minreg will ignore combinations of co-regulators AR $\cup \{r_1, r_2\}$ if the marginal score values of AR $\cup \{r_1\}$ and AR $\cup \{r_2\}$ are both low, although AR $\cup \{r_1, r_2\}$ may be significant. In such a case, $r_1$ and $r_2$ are said to cooperate (*i.e.*, they should act collectively in order to influence their target gene(s)). Note that very few computational approaches have investigated the role of regulator cooperativity [19, 31], although such mechanisms have been identified in many organisms (*Saccharomyces cerevisiae, Homo sapiens, etc.*).

***Module Networks.*** Reference [59] exploits the idea that in many large domains, variables can be partitioned into sets so that variables of each set have similar parent sets. In gene regulatory modeling, this assumption reflects the well-founded idea that some sets of genes are co-regulated by the same inputs in order to ensure the coordination of their joint activity. In the module network framework, a module $M_j$ is defined by a set of parents Pa[$M_j$] and a conditional probability template (CPT) P($M_j$—Pa[$M_j$]) that specifies a distribution over the values that the variables in the modules can take for each assignment of the parent's values.

The "parameters" of the model are now the number of modules, the dependency structure over the modules, and the conditional probability template for each module. A nonparametric tool, namely regression trees [66], has been used here to model these conditional distributions, allowing a great flexibility for dependencies. Such a modeling has appeared to be relevant when the network involves regulatory cascades with large groups of co-regulated genes, for instance, in the cell cycle of the yeast.

## 41.7 CONSTRAINT-BASED DATA MINING

The problem of computing the set of frequent itemsets and the associated set of association rules has been introduced in the seminal paper of Agrawal [1], in the context of the problem of "market basket analysis": Given a set of transactions, formed by lists of items bought together, the goal is to identify itemsets frequently bought together.

The task of finding frequent itemsets can be modeled as follows. An extraction context is a triplet $(\mathcal{O}, \mathcal{A}, \mathcal{B})$ such that $\mathcal{O}$ is a finite set of observations, $\mathcal{A}$ is a finite set of attributes (items) and $\mathcal{B}$ is a binary relationship between $\mathcal{O}$ and $\mathcal{A}$, $(\mathcal{B} \subseteq \mathcal{O} \times \mathcal{A})$. $\mathcal{B}$ will also be refered to as an extraction context in the following. An *itemset* is a set of items or attributes of $\mathcal{A}$. An intemset of size $k$ is refered to as a $k$-itemset. The set of itemsets $L = 2^{\mathcal{A}}$ is a lattice, partially ordered by the inclusion relationship denoted $\subseteq$ [46]. The support of itemset $m$, denoted Supp($m$), is the proportion of observations in $\mathcal{B}$ containing $m$ : $\text{supp}(m) = \frac{|\{(o \in \mathcal{O}|m \subseteq o\}|}{|\mathcal{O}|}$. The min-support constraint is defined as follows: Given a user defined threshold $\gamma$, named minimal support, an itemset $m$ is frequent in a context $\mathcal{B}$, if $\text{supp}(m) \geq \gamma$.

Constraint-based mining selects all itemsets of $L$ satisfying a property $q$, referred to as a constraint hereafter. A constraint formalizes the interest of the user to focus on the most promising patterns according to his point of view. Search for itemsets of $L$ satisfying a constraint $q$ is in the worst-case exponential in the size of $\mathcal{A}$. However, it can be optimized if $q$ has some properties, such as *anti-monotonicity*. A constraint $q$ is anti-monotonic with respect to $\subseteq$, If for all $m$ and $m'$ itemsets of $L$ such that $m \subseteq m'$, if $q(m')$ is satisfied, then $q(m)$ is also satisfied.

This anti-monotonicity property allows to derive the following safe pruning conditions when exploring $L$ for itemsets satisfying an anti-monotonic property $q$: If an itemset does not satisfy an anti-monotonic property $q$, none of its supersets will ever satisfy $q$. The min-support constraint mentioned above is anti-monotonic wrt $\subseteq$: If an itemset $m$ is frequent, all its subsets are necessarily frequent. In the example of Figure 41.3, DE is a minimal itemset that does not satisfy the min-support constraint. As a consequence, none of its supersets need to be generated and evaluated. Note that some works [51] replace the frequency by other interestingness measures to evaluate the relevance of itemsets.

APriori [1], given a user supplied min-supp threshold, looks for every itemset frequent in a transactional dataset. This is a level-wise "generate and test" algorithm:

| $\mathcal{O}$ | **Transaction** |
|---|---|
| $o_1$ | A B C D E |
| $o_2$ | A B C E |
| $o_3$ | C |
| $o_4$ | B C |
| $o_5$ | A B C D |
| $o_6$ | A B C |

| $\mathcal{O} \backslash \mathcal{A}$ | A | B | C | D | E |
|---|---|---|---|---|---|
| $o_1$ | 1 | 1 | 1 | 1 | 1 |
| $o_2$ | 1 | 1 | 1 | 0 | 1 |
| $o_3$ | 0 | 0 | 1 | 0 | 0 |
| $o_4$ | 0 | 1 | 1 | 0 | 0 |
| $o_5$ | 1 | 1 | 1 | 1 | 0 |
| $o_6$ | 1 | 1 | 1 | 0 | 0 |

**Figure 41.3** Two representations of a transactional dataset $\mathcal{B}$.

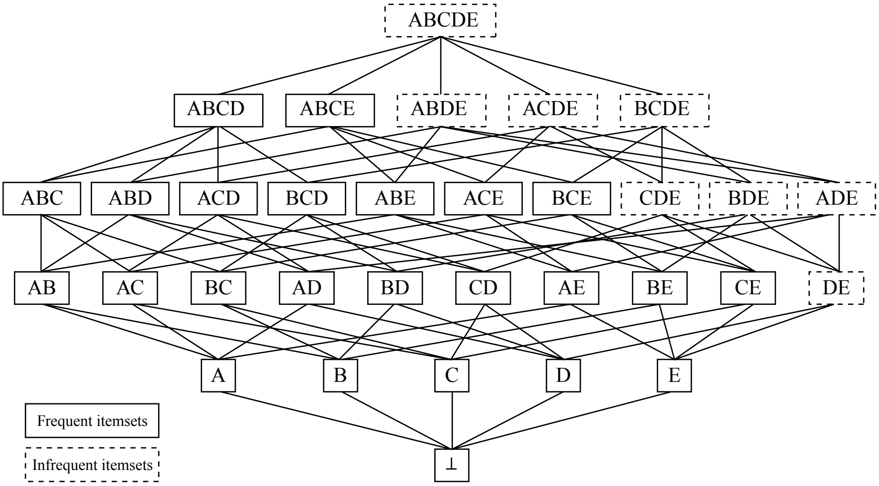


**Figure 41.4** Lattice of frequent motifs from $\mathcal{B}$ (see Figure 41.3) with the min-support threshold $\gamma = 2$.

At iteration $k$, it generates all candidate k-itemsets, using frequent $(k - 1)-$itemsets found during previous iteration. Then, exact frequency of candidate $k$-itemsets is evaluated with a dataset scan. For instance, given the dataset $\mathcal{B}$ of Figure 41.3, APriori with a min-supp threshold of $2/6$ computes 23 frequent itemsets (see Figure 41.4).

Although the worst-case complexity of APriori is exponential in the number of attributes of $\mathcal{A}$, it may be quite efficient if the data analyzed is sparse. Its empirical efficiency relies on specific data structures structures (hash-tree, trie or FP-tree)—see [38] for a review. Other search strategies in the lattice of itemsets of $\mathcal{A}$ have been implemented, such as depth-first with backtrack [12], or best-first search [36].

When the itemsets satisfying a set of constraints are too many to be extracted efficiently, many algorithms build approximate or condensed representations [16] for candidate itemsets. The most studied condensed representation is that of closed patterns, first introduced in [53]. The closure operator $f$ associates with an itemset $m$ the maximal set of items common to all the observations containing $m$: The closure of an itemset $m$ is equal to the intersection of all the observations containing it. For itemsets, the closed motif corresponding to a motif $m$ is the largest superset of $m$ with the same support set. Extraction of closed itemsets may show a high-performance increase especially when data are highly correlated [38].

### 41.7.1 Multiple Usages of Extracted Patterns

Extracted motifs may be used for very different purposes. They can be used as such, as a special type of *local information*, or they can be combined with each other to build a global model, descriptive or predictive [24]. Local patterns encode knowledge that is out of the scope of classical statistical analyses (*e.g.*, multivariate analysis), which tend to ignore low-frequency events. Deriving association rules [1] is one of the most frequent use of local patterns. Association rules capture local correlations observed in data. An association rule built on a pattern $Z$ is an implication of the form : $X \rightarrow Y$, where $X \subset Z$ and $Y = X \setminus Z$. $X$ is called the premise or the left-hand side of the rule, whereas $Y$ is the conclusion or right-hand side of the rule. Association rule algorithms usually look for rules satisfying a minimal support and confidence, or other syntactic properties such as size of its premise. The support of rule $X \rightarrow Y$ is the support of $X \cup Y$, and its confidence is its support divided by the support of its premises, and denotes the conditional probability that the conclusion of the rule is satisfied when its premise is. Note that the confidence measure is neither monotonic nor anti-monotonic, which excludes easy pruning the search space of candidate association rules. Rules with low support may well have a high confidence and be quite useful from an application viewpoint [48]. Local patterns may also be combined to obtain global models (such as top-n patterns [24]) in order to reduce the extracted patterns to the most “significant” ones, yielding an improved resistance to noise in the data.

### 41.7.2 Mining Gene Regulation from Transcriptome Datasets

Frequent and constrained data mining algorithms have been applied to publicly available transcriptome datasets from various organisms. To fit with the previously introduced framework, an item is a gene/transcription factor, a transaction is a condition/observation, and frequent itemsets are co-regulated genes. Such techniques have at least two major advantages with respect to classical statistical clustering techniques. First, itemset mining techniques are able to build overlapping gene groups (*i.e.*, a gene that functions in numerous physiological pathways may be associated to several groups), without limitations. Second, they allow the discovery of expression patterns observed in a subset only of all the conditions examined, sometimes of relatively low support.

Whatever organism tackled, expression datasets share three characteristics that makes them quite different from market basket analysis datasets.

First, transcriptome data are continuous and therefore usually have to be discretized before being handled by a data mining algorithm[1]. Several techniques for discretizing continuous datasets have been studied [29], some general ones and some application-oriented ones. In the context of gene expression mining, we refer to [56] that propose several application-oriented techniques as well as a technique to quantify information loss due to this reformulation step. Most often, gene expression datasets are binarized. For instance, 1 means the gene is differentially expressed in the observed conditions, and 0 means that it is not. Some approaches [31] handle three-valued gene-expression datasets, which allows the representation of finer-grained gene variation (over-, normal, and under-expression). Discretization can smooth the stochastic variations of gene expression and be a partial solution to overfitting.

[1] Notice some association rule mining may handle data directly in a numeric format [37].

Second, the number of attributes/items is very large with respect to the number of observations/conditions. As a consequence, the search space for frequent co-expressed genes (*i.e.*, the lattice $L$), is huge and an exhaustive level-wise type of search is intractable unless the dataset is sparse, especially when mining constraints are loose (support, number of gene in a co-expression cluster, *etc.*). As the complexity of data mining algorithms is in the worst-case exponential with the number of attributes/items, mining and biological expertise is used to reduce the number of items considered.

Finally, noise, both experimental and by-product of diverse preprocessing steps (normalization, discretization, *etc.*) occurs in the data, with the effects of increasing the size of results, making it even more difficult to detect knowledge nuggets among the results [44]. We will point out specific works in the following that address at least one of these issues in a gene expression mining context.

***Constrained Itemsets.*** Constrained itemset mining approaches try to establish synexpression groups, that is, groups of genes whose expression is correlated in different biological situations. The system described in [40] extracts gene sets that are positively and negatively correlated for three-valued datasets. In [52, 11], the authors proposed to extract *formal concepts* (*i.e.*, associations of closed gene sets and closed conditions sets from gene expression data). These formal concepts are interesting for biological interpretation because they can be seen as candidate transcription modules [10]. They build closed gene sets by manipulating a transposed version of the expression context and by enumerating condition sets rather than gene sets, which drastically improves the efficiency of the extraction method when the number of conditions is small with respect to the number of genes. Besson *et al.* [9] propose the D-miner algorithm, which is a general-purpose mining algorithm, dedicated to the extraction of constrained bi-sets or formal concepts. This system is used to enrich the extraction context with boolean information coding the association of transcription factors with genes, when the information is available. This allows inferring more relevant concepts, associating gene with both biological conditions and transcription factors. Similarly, [69] looks for formal concepts observed in a number of different datasets.

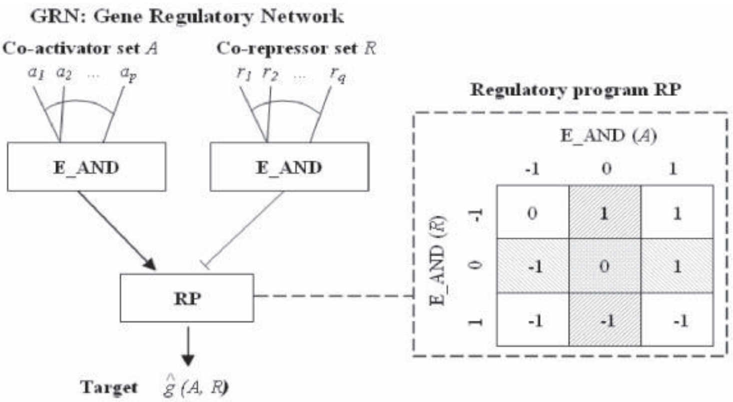
***Association Rules.*** Association Rules Discovery has the goal of identifying sets of genes whose expression is correlated [6, 23, 57, 22]. The kind of association rules discovered is of the following form: *When gene A and gene B are expressed, then gene C is often expressed.* Although association rules encode that the expression of gene sets are correlated, this is clearly not sufficient to capture the complexity of a regulation network. Still, let us mention here some successful attempts to extract some local structure from gene expression datasets. Reference [6] uses the notion of *free* itemsets in the context of SAGE expression analysis. An itemset $m$ is free if none of its subsets are linked within a strong association rule (*i.e.*, a rule with confidence 100%). Instead of finding individual association rules, FARMER [22] finds interesting rule groups that are essentially a set of rules that are generated from the same set of rows (conditions). Unlike conventional rule mining algorithms, FARMER searches for interesting rules in the row enumeration space and exploits all user-specified constraints, including minimum support, confidence, and chi-square to support efficient pruning.

Unless very strict constraints are set on the expected association rules, the number of rules obtained is most of the time very high [23], which makes the interpretation task of biologists quite difficult. Some authors chose to enrich gene expression datasets with external data sources in the aim of selecting relevant association rules (*i.e.*, *Gene ontology* [17] or data about bindings DNA-proteins (*ChIP-chip*) [57]).

***LICORN (LearnIng CoOperative Regulation Networks).*** Elati *et al.* [31] have recently proposed an original, scalable technique called LICORN for deriving cooperative regulations, in which many co-regulators act together to activate or repress a target gene. Let us denote by $\mathcal{R}$ the set of genes with a known or putative regulation activity and $\mathcal{G}$ as the set of genes without such an activity. The input of the mining method is a discretized expression matrix for genes of $\mathcal{R} \cup \mathcal{G}$. Each expression value can take the value $-1$ (under-expressed), 0 (normal), or 1 (over-expressed). A gene regulatory network (GRN) associated with a target gene $g$ is a pair $(A, I)$, where $A \subseteq \mathcal{R}$ is a co-activator set, and $I \subseteq \mathcal{R}$ is a co-inhibitor set. The set of GRNs for all target genes can also be seen as a bipartite graph where the top layer contains regulators, the bottom layer contains target genes, and edges code for a regulatory interaction between regulators and target genes, each edge being labeled with a regulatory mode (*i.e.*, *activator* or *inhibitor*). The regulation relations of interest here are combinatorial: Each target gene has a number of activators and/or inhibitors. Activators on the one side and inhibitors on the other side are aggregated in the model through an extended logical ANDs (*i.e.*, a regulator set $S$ (activator or repressor) is over-expressed (respectively, under-expressed) if and only if all the regulators in $S$ are over-expressed (respectively, under-expressed)). Finally, the Figure 41.5 describe a discrete function called *Regulatory Program* (RP), which, given the combined states of activators $A$ and inhibitors $I$ of $g$ in a sample $s$ computes $\hat{g}_s(A, I)$, the estimated state of $g$ in $s$. The main features of this regulation model are therefore the explicit representation of activation and repression relationships for a given target gene, and the representation of cooperative transcriptional regulation.

LICORN uses an original heuristic approach to accelerate the search for an appropriate structure for the regulation network. It first computes frequent co-regulator sets (*i.e.*, regulator sets that frequently occur together as over (1) or under (-1)-expressed in the discretized expression matrix). This is done by using an extension of the Apriori algorithm [1] to handle both 1 and -1 supports[2]. From this representation, a limited subset of candidate co-regulator sets is then associated with each gene. The learning algorithm looks for each gene for regulator sets, which have a high "overlap" with the target gene. Intuitively, the overlap constraint checks the size of the intersection between supports of the target gene and a given candidate co-regulator set. A candidate activator set for a target gene $g$ is frequently over-expressed when $g$

[2] The x-support of a co-regulator $C$ in the three-valued expression matrix is the set of samples that include all the regulators of $C$ with the state $x$.



| E_AND ($R$) \ E_AND ($A$) | -1 | 0 | 1 |
|---|---|---|---|
| -1 | 0 | 1 | 1 |
| 0 | -1 | 0 | 1 |
| 1 | -1 | -1 | -1 |

**Figure 41.5** Definition of the regulatory program RP, which interprets as follows: i) GRN contains coactivators only; RP checks the correlation between the status of the target gene and that of its regulators. ii) GRN contains coinhibitors only; RP checks the anticorrelation between the status of the target gene and that of its regulators. iii) GRN contains both coactivators and coinhibitors; this regulation program specifies a combinatorial interaction. For example, the target gene is over-expressed only when the coactivators are over-expressed and the coinhibitors are not.

is over-expressed or frequently under-expressed when *g* is under-expressed. On the opposite, a candidate repressor set for a target gene *g* is frequently over-expressed when *g* is under-expressed and vice-versa. This search can be efficiently performed because of the property of anti-monotonicity of the overlap constraint with respect to set inclusion. Then, once a limited number of candidate activator and inhibitor sets have been obtained, an exhaustive search for the best gene regulatory network can be performed. Finally, a permutation-based procedure is used for selecting statistically significant regulation relations. Authors have shown in [31] that the cooperative regulation patterns inferred by LICORN cannot be identified by clustering or pairwise methods, and are only partly revealed by constrained Bayesian or decision tree-based techniques, such as those used in previous studies [55, 59].

## 41.8 VALIDATION

Within the scope of network inference, the validation can essentially take two forms: First the results of the learning algorithm have to be assessed by some quality measure in order to inform the biologist about the success or the failure of the inference process. Second, if the learned model is considered as relevant according to the criteria of statistical machine learning, it is mandatory to confront the model to new sources of knowledge or data to confirm or infirm each piece of inferred information. In the following, we will call the first kind of validation statistical validation and the second one, biological validation. Statistical validation associates some confidence scores to the links inferred in order to assess their likelihood, given the current model, and also evaluate the robustness of the inference procedures to data variance. To implement the biological validation step, the biologists and the computer scientist/statistician can re-design appropriate additional experiments targeted toward the test of the most relevant extracted pathways or they can screen the literature and mine the existing biological databases for testing the validity of the hypothesis using independent sources of information. Alternately, most of the authors evaluate their inference tool by exploited public data related to known networks in order to show the quality of their approaches and present also extensive results on artificial data generated from artificial networks, allowing an empirical study of the algorithm behavior.

### 41.8.1 Statistical Validation of Network Inference

All algorithms described in Sections 41.5, 41.6, and 41.7 produce a final hypothesis or a set of hypotheses. To complete the procedure of reverse modeling, a process able to reject or accept the proposed modeling must be defined and applied. Note that the produced hypothesis may be interesting regarding a subset of the involved variables only and yet bring local but valuable information about the network. We present in the following three methods that contribute to assess the quality of the produced network.

***Model Selection.*** Statistical estimation methods such as a bootstrap offer a convenient means to measure the sensitivity of some quality criterion to data variance, especially when the posterior distribution $\Pr(D|M)$, where $D$ is the data observed and $M$ is the inferred network, is unknown or difficult to estimate. Once a quality criterion is fixed, a boostrap can be used to estimate the variance of this criterion computed when the learning sample varies. This procedure allows are to measure to what extent the algorithm provides a robust model. A boostrap is frequently used to select a hypothesis among a set of candidate hypotheses: In machine learning, this is the most general way to do model selection because no information about the background distribution is needed. This method has been used, for instance by [31], that infers cooperative regulators GRN for a set of genes, in order to evaluate how unusually low the score of the best GRN is with respect to the scores that would have been observed if there was no biological relationship between regulators and target gene expression. The absence of a biological relationship between the target and candidate regulators in the GRN is checked, using random permutations of the samples in the gene expression matrix. The problem of this approach is that, as thousands of genes are tested simultaneously, we must then expect a high number of false-positives among the networks retained. As a means of overcoming this problem, [7] suggested the selection of a threshold making it possible to control the false discovery rate (FDR), which corresponds to the expected proportion of false-positives among selected GRNs. As the procedure they propose has only been shown to control the FDR when the tests are independent, which is far from true for gene expression, the more conservative procedure proposed by [8] can be used, which gives strong control of the FDR for any dependence structure.

***Cross-validation.*** One of the well-known difficulties for evaluating unsupervised methods in the context of regulation network inference is that there is no objective criterion for assessing what a “good” interaction between two genes is [68]. One usual way to proceed, once the network has been inferred, is to transform the problem into a supervised learning one, in order to assess the inferred networks with the range of estimation tools provided in a supervised learning context. The network can then be seen as a classifier, able to predict the gene expression level of its target gene, given the state of its regulators. For a prediction measure to be objective, it must be evaluated on a validation set that has not been used to build the predictor. Cross-validation is widely used in classification and learning theory. This approach involves partitioning the observed population $\mathcal{S}$ into $K$ subgroups $\mathcal{S}_1, \ldots \mathcal{S}_K$. For $k = 1 \ldots K$, the predictor is built on the *training population* $\mathcal{S} \setminus \mathcal{S}_k$, and its performance is evaluated on the *test population* $\mathcal{S}_k$. In practice, *ten-fold cross-validation* is often considered, as this method provides a fair estimate of the prediction error at a reasonable computational cost (10 training runs with $\frac{|\mathcal{S}|}{10}$ observations each). When the dataset size is very limited, it is possible to use a single test example at each validation step and to repeat $\mid \mathcal{S} \mid$ the validation step, this method is known as *leave-one-out*.

Authors of [31] for instance use a 10-fold cross validation to evaluate Licorn’s performance. The authors used MAE (Mean Average Error) as a measure of the discrepancy between actual gene expression and gene expression inferred from the activity of regulators through its GRN. Peer *et al.* [55] perform a 5-fold cross-validation to study the performance of their MINREG system, assessing the capability of inferred regulators to predict the gene expression level of their target gene. They have shown that MINREG performs better than co-expression gene networks. LICORN significantly outperformed MINREG on two yeast datasets ($p$-value in paired $T$-tests of $1.6 \times 10^{-8}$ for the Spellman dataset and $6.7 \times 10^{-9}$ for the Gasch dataset).

***Prediction Performance on a Known Network.*** When learning algorithms are applied to known networks (real or simulated), it is possible to evaluate the results of the learning process in term of class prediction. The concept of regulation from one gene (regulator) to another gene (target gene) is used as the class to be predicted. The quality of a network inference system can be evaluated by measuring the number of true positive regulations (TP), the number of false-positive regulations (FP), and the number of true negative regulations of this class (TN). Until now, we have described an evaluation method that only takes into account a single measure: empirical error. It can be useful, in a supervised context, to take into account finer grained measures, such as the False-Positive and False-Negative rate. Indeed the costs of a false-positive and a false-negative are usually not identical, and a higher error rate can be preferred if it allows to reduce the highest cost errors. $ROC$ curves (*Receiver Operating Characteristic*) make it possible to tune this compromise [13]. However, using ROC curves in order to evaluate the inferred networks implies that the real underlying regulation network is available, which is not the case for most organisms. As a consequence, the false-positive rate cannot be evaluated. The recent trend is to use synthetic datasets obtained by simulating an artificial network [60] in order to systematically assess the robustness of the inference algorithm to environmental and internal parameters variations (*e.g.*, noise, size of the training set, *etc.*). Recently, [27] proposes a generator, SynTReN, that takes into account biological knowledge, and, in particular, network topology. However, evaluation based on artificial datasets are optimistic because the same model is used both for generating datasets and for infering the network.

### 41.8.2 Biological Validation

Network inference aims at helping the biologists to discover a new pathway and refine existing ones. There are many ways to biologically validate infered networks. The network infered is usually of a large size; in that case, post-processing of the results has to take place for extracting salient information to be proposed to the biologists. Many biologically pertinent questions about gene regulation and networks have direct counterparts in graph theory and can be answered using well-established methods and algorithms on graphs. For example, one may wish to identify highly interacting genes, resolve cascades of gene activity, compare gene networks (or pathways) for finding high degree vertices, topological vertex ordering, and graph iso(homo)-morphisms. For instance, analysis of the structure and organization of networks inferred by LICORN revealed several notable features. In both stress response GRNs (Gasch dataset) and cell cycle GRNs (Spellman dataset), the authors have studied the number of interactions learned, the number of regulators/target genes, and vice versa. They have found that the distribution of the out-going connectivity is best approximated by a power-law equation. This allowed us to detect regulator hubs [42] with high out-going connectivity (*e.g.*, the heat shock and osmolarity stress regulator *PPT1* regulates 300 target genes). Note that these cumulatively account for more than 60% of the interactions in both datasets. For most regulators in both datasets, a linear dependence was observed between the number of target genes regulated by a given regulator and the number of co-regulators of that regulator. However, regulators from the Spellman dataset have a higher number of co-regulators than do regulators from the Gasch dataset, indicating that there are more cooperative associations between regulators in cell cycle GRNs. All these results are consistent with recent advances [39, 45, 3] concerning the characterization of topological transcriptional network features in yeast and provide the first evidence of the relevance of inferred GRNs.

Some research teams have tried to cross-check the infered networks with additional sources of information. As far as yeast is concerned, a large amount of biological knowledge is available: functional information, in the *Saccharomyces Genome Database* [20], documented regulations in the *YEASTRACT* database [65], protein-protein interactions in the *BioGRID* database [63]. Thus, the transcriptional networks identified for yeast can be checked by comparison with various sources of information. As an example, Segal *et al.* [59] exploit their module network approach for the yeast cell cycle data, starting from a set of regulators candidates containing both putative and known transcription factors. First, biological validation consists of scoring the functional coherence of each module according to its gene covered by annotations significantly enriched in the modules. Second, they test whether their method correctly predicts targets of each regulator by analyzing the distribution of gene differentially expressed in modules. Third, they try to identify the process regulated by each regulator and thus corresponding to the modules. They selected three hypotheses suggested by the method, involving uncharacterized putative regulators and processed the relevant yeast deletion strains. Altogether, two of the three regulators were confirmed by the various additional sources of information including the new experiments. Reference [31] evaluated the cooperativity between the coregulators inferred by LICORN, based on two assumptions: (i) the existence of protein-protein interactions between coregulators implies participation in the same regulation mechanism, and (ii) targets contributing to a similar biological process are regulated by the same control mechanism.

Although most attempt to unraveling gene regulatory networks have focused on yeast, we can notice the methods have begun to be applied to higher eukaryotes such as mouse or human. Reference [5] used the system ARACNE for the reconstruction of regulatory networks from expression profiles of human B cells. Their results suggested a hierarchical, scale-free network where a few highly interconnected genes (hubs) account for most of the interactions. Confrontation of the inferred network against available data led to the identification of MYC as an important hub, which controls known target genes as well as new ones, which were biochemically validated. The newly identified MYC targets include some major hubs. This approach can be generally useful for the analysis of normal and pathology networks in mammalian cells.

## 41.9 CONCLUSION AND PERSPECTIVES

To understand most cellular processes, one must understand how genetic information is processed. A formidable challenge is the dissection of gene regulatory networks to delineate how eukaryote cells coordinate and govern patterns of gene expression that ultimately lead to a phenotype. In this chapter, we have reviewed several approaches for modeling gene regulatory networks and for reverse engineering such networks from experimental observations. We have discussed three important unsupervised learning models: co-expression network, Bayesian networks, and constrained item set mining. This review attempted to highlight some of the methodological advantages brought by machine learning as well as the difficult technical points yet to be solved. Nevertheless, a first generation of tools have demonstrated on problems of limited complexity and on biological case studies that network inference is possible but requires prior knowledge integration and dimension reduction to be scalable. Regarding these last points, unsupervised learning techniques have appeared as a promising framework, supporting the management of uncertainty, the estimation of hidden variables and modular approaches. From the literature, it appears that the most interesting machine learning works are those that have been carefully validated, both statistically and biologically. This suggests that in an ideal setting, the computer scientists and the statisticians should be involved in the discovery process from the beginning (*i.e.*, from the experiment design to the biological validation of the discovered hypotheses). Regarding this modeling and discovery process, there is a large, open field for various learning applications and complex data analysis. We therefore think that the use of machine learning in the process of discovery in system biology is only at its infancy.

For this approach to be fully successful, several issues must be addressed. First, such machine learning methods should exploit the wide knowledge stored in existing databases in order to be able to confront these sources with experimental data from the lab. Second, if network inference has to take place at the scale of a whole genome, the detailed models must be replaced by hierarchical and modular approaches that reduce the dimension of the search space. Third, integrative views that combine for instance genetic networks, protein-protein interaction networks, and metabolic networks have not yet been worked out and must be considered. Finally, active learning methods can be explored to suggest which interventions (*i.e.*, knockouts or overexpressions) should be performed in order to increase our knowledge about the gene network structure.

## REFERENCES

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proceedings of the International Conference on Management of Data*, 1993, pp. 207–216.
2. E. Alm and A.P. Arkin. Biological networks. *Curr Opin Struct Biol*, 13:193–202, 2003.
3. S. Balaji, M.M. Babu, L.M. Iyer, N.M. Luscombe, and L. Aravind. Comprehensive analysis of combinatorial regulation using the transcriptional regulatory network of yeast. *J Mol Biol*, 360:204–212, 2006.
4. A.L. Barabasi and Z.N. Oltvai. Network biology: Understanding the cell's functional organization. *Nat Rev Genet*, 5:101–113, 2004.
5. K. Basso, A.A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano. Reverse engineering of regulatory networks in human B cells. *Nat Genet*, 37:382–390, 2005.
6. C. Becquet, S. Blachon, B. Jeudy, J.F. Boulicaut, and O. Gandrillon. Strong-association-rule mining for large-scale gene-expression data analysis: A case study on human sage data. *Genome Biol*, 3: research0067.1–research0067.16, 2002.
7. Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *J Roy Stat Soc*, 57:289–300, 1995.
8. Y. Benjamini and Y. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Ann Stat*, 29:1165–1198, 2001.
9. J. Besson, C. Robardet, J.F. Boulicaut, and S. Rome. Constraint-based concept mining and its application to microarray data analysis. *Intell Data Anal*, 9:59–82, 2005.
10. E. Birmele, M. Elati, C. Rouveirol, and Ch. Ambroise. Identification of functional modules based on transcriptional regulation structure. *BMC Proceedings*, 2:S4, 2008.
11. S. Blachon, R.G. Pensa, J. Besson, C. Robardet, J.F. Boulicaut, and O. Gandrillon. Clustering formal concepts to discover biologically relevant knowledge from gene expression data. *In Silico Biol*, 7:467–483, 2007.

12. C. Borgelt. Efficient implementations of apriori and eclat. *1st Workshop of Frequent Item Set Mining Implementations*, 2003.

13. A.P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recogn*, 30:1145–1159, 1997.

14. P. Brazhnik, A. de la Fuente, and P. Mendes. Gene networks: How to put the function in genomics. *Trends Biotechnol*, 20:467–472, 2002.

15. A.J. Butte and I.S. Kohane. Mutual information relevance networks: Functional genomic clustering using pairwise entropy measurements. *Pacific Symposium in Biocomputing*, 2000, pp. 418–429.

16. T. Calders, C. Rigotti, and J.F. Boulicaut. A survey on condensed representations for frequent sets. *Constraint-Based Mining and Inductive Databases*, volume 3848, Springer, New York, 2004, pp. 64–80.

17. P. Carmona-Saez, M. Chagoyen, A. Rodriguez, O. Trelles, J. Carazo, and A. Pascual-Montano. Integrated analysis of gene expression by association rules discovery. *BMC Bioinformatics*, 7:54, 2006.

18. S.L. Carter, C.M. Brechbuhler, M. Griffin, and A.T. Bond. Gene co-expression network topology provides a framework for molecular characterization of cellular state. *Bioinformatics*, 20:2242–2250, 2004.

19. Y.H. Chang, Y.C. Wang, and B.S. Chen. Identification of transcription factor cooperativity via stochastic system model. *Bioinformatics*, 22:2276–2282, 2006.

20. J.M. Cherry, C. Adler, C. Ball, S.A. Chervitz, S.S. Dwight, E.T. Hester, Y. Jia, G. Juvik, T. Roe, M. Schroeder, S. Weng, and D. Botstein. SGD: Saccharomyces Genome Database. *Nucleic Acids Res*, 26:73–79, 1998.

21. D.M. Chickering. Optimal structure identification with greedy search. *J Mach Learn Res*, 3:507–554, 2002.

22. G. Cong, A.K.H. Tung, X. Xu, F. Pan, and J. Yang. FARMER: Finding interesting rule groups in microarray datasets. *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, Paris, France, 2004, pp. 143–154.

23. C. Creighton and S. Hanash. Mining gene expression databases for association rules. *Bioinformatics*, 19:79–86, 2003.

24. B. Crémilleux and A. Soulet. Discovering knowledge from local patterns with global constraints. *Proceedings of the International Conference on Computational Science and Its Applications, Part II*, Perugia, Italy, Springer-Verlag, New York, 2008, pp. 1242–1257.

25. F. D'Alché Buc. Inference of biological regulatory networks: Machine learning approaches. In François Képès, editor, *Biological Networks*, World Scientific, Singapore, 2008, pp. 41–83.

26. A. de la Fuente, N. Bing, I. Hoeschele, and P. Mendes. Discovery of meaningful associations in genomic data using partial correlation coefficients. *Bioinformatics*, 20:3565–3574, 2004.

27. T. Van den Bulcke, K. Van Leemput, B. Naudts, P. van Remortel, H. Ma, A. Verschoren, B. De Moor, and K. Marchal. SynTReN: A generator of synthetic gene expression data for design and analysis of structure learning algorithms. *BMC Bioinformatics*, 7:43, 2006.

28. P. D'haeseleer, S. Liang, and R. Somogyi. Genetic network inference: From co-expression clustering to reverse engineering. *Bioinformatics*, 16:707–726, 2000.

29. J. Dougherty, R. Kohavi, and A. Sahami. Supervised and unsupervised discretization of continuous features. *Machine Learning: Proceedings of the Twelfth International Conference*, 1995, pp. 487–499.

30. S. Draghici. *Data Analysis Tools for DNA Microarrays*. Chapman & Hall/CRC, Boca Raton, FL, 2003.

31. M. Elati, P. Neuvial, M. Bolotin-Fukuhara, E. Barillot, F. Radvanyi, and C. Rouveirol. LICORN: Learning cooperative regulation networks from gene expression data. *Bioinformatics*, 23:2407–2414, 2007.

32. L. Elnitski, V.X. Jin, P.J. Farnham, and S.J.M. Jones. Locating mammalian transcription factor binding sites: A survey of computational and experimental techniques. *Genome Res*, 16(12):1455–1464, 2006.

33. J. Faith, B. Hayete, J.T. Thaden, I. Mogno, J. Wierzbowski, G. Cottarel, S. Kasif, J.J. Collins, and T.S. Gardner. Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles. *PLoS Biol*, 5:e8, 2007.

34. N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303:799–805, 2004.

35. N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian network to analyze expression data. *Comput Biol*, 7:601–620, 2000.

36. A.W. Fu, R.W. Kwong, and J. Tang. Mining n-most interesting itemsets. *Proceedings of the 12th International Symposium Foundations Intelligent Systems (ISMIS)*, Springer-Verlag, New York, 2000, pp. 59–67.

37. E. Georgii, L. Richter, U. Ruckert, and S. Kramer. Analyzing microarray data using quantitative association rules. *Bioinformatics*, 21:123–129, 2005.

38. B. Goethals and M.J. Zaki. Advances in frequent itemset mining implementations: Introduction to FIMI03. *FIMI*, volume 90, 2003.

39. N. Guelzim, S. Bottani, P. Bourgine, and F. Képès. Topological and causal structure of the yeast transcriptional regulatory network. *Nat Genet*, 31:60–63, 2002.

40. L. Ji and K. L. Tan. Mining gene expression data for positive and negative co-regulated gene clusters. *Bioinformatics*, 20:2711–2718, 2004.

41. L. Ji and K. L. Tan. Identifying time-lagged gene clusters using gene expression data. *Bioinformatics*, 21:509–516, 2005.

42. T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, G.K. Gerber, N.M. Hannett, C.T. Harbison, C.M. Thompson, I. Simon, J. Zeitlinger, E.G. Jennings, H.L. Murray, D.B. Gordon, B. Ren, J.J. Wyrick, J.B. Tagne, T.L. Volkert, E. Fraenkel, D.K. Gifford, and R.A. Young. Transcriptional regulatory networks in Saccharomyces cerevisiae. *Science*, 298:799–804, 2002.

43. W.P. Lee and W.S. Tzou. Computational methods for discovering gene networks from expression data. *Brief Bioinformatics*, 10:408–423, 2009.

44. J. Liu, S. Paulsen, W. Wang, A. Nobel, and J. Prins. Mining approximate frequent itemsets from noisy data. *Proceedings of the Fifth IEEE International Conference on Data Mining*, IEEE Computer Society, Piscataway, NJ, 2005, 721–724.

45. N.M. Luscombe, M.M. Babu, H. Yu, M. Snyder, S.A. Teichmann, and M. Gerstein. Genomic analysis of regulatory network dynamics reveals large topological changes. *Nature*, 431:308–312, 2004.

46. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Min Knowl Discov J*, 1:241–258, 1997.

47. F. Markowetz and R. Spang. Inferring cellular networks—a review. *BMC Bioinformatics*, 8 (Suppl 6):S5, 2007.

48. S. Chawla and T. McIntosh. High confidence rule mining for microarray analysis. *IEEE/ACM Trans Comput Biol Bioinform*, 4(4):611–623, 2007.
49. M. Middendorf, A. Kundaje, C. Wiggins, Y. Freund, and C. Leslie. Predicting genetic regulatory response using classification. *Bioinformatics*, 20:232–240, 2004.
50. T.M. Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.
51. R.T. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. *SIGMOD Rec*, 27(2):13–24, 1998.
52. F. Pan, G. Cong, A.K.H. Tung, J. Yang, and M.J. Zaki. Carpenter: Finding closed patterns in long biological datasets. *KDD '03: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 637–642.
53. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Pruning closed itemset lattices for association rules. *Actes 14ème Conférence Bases de Donnes Avances (BDA'98)*, 1998, pp. 177–196.
54. J. Pearl. On the connection between the complexity and credibility of inferred models. *Int J Gen Syst*, 4:255–264, 1978.
55. D. Pe'er, A. Regev, and A. Tanay. Minreg: Inferring an active regulator set. *Bioinformatics*, 18:258–267, 2002.
56. R.G. Pensa, C. Leschi, J. Besson, and J.F. Boulicaut. Assessment of discretization techniques for relevant pattern discovery from gene expression data. *Proceedings of the 4th ACM SIGKDD Workshop on Data Mining in Bioinformatics BIOKDD'04*, 2004, pp. 24–30.
57. T.H. Pham, K. Satou, and T.B. Ho. Mining yeast transcriptional regulatory modules from factor dna-binding sites and gene expression data. *Genome Inform*, 15:287–295, 2004.
58. M. Schena, D. Shalon, R. Davis, and P. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 70:467–470, 1995.
59. E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: Identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34:166–176, 2003.
60. V.A. Smith, E.D. Jarvis, and A.J. Hartemink. Evaluating functional network inference using simulations of complex biological systems. *Bioinformatics*, 18:216S–224S, 2002.
61. R. Somogyi and H. Kitano. Gene expression and genetic networks. *Pacific Symposium in Biocomputing*, 1999, pp. 3–4.
62. P. Spirtes, C. Glymour, R. Scheines, S. Kauffman, V. Aimale, and F. Wimberly. Constructing Bayesian network models of gene expression networks from microarray data. *Proceedings of the Atlantic Symposium on Computational Biology, Genome Information Systems and Technology*, 2000.
63. C. Stark, B.J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: A general repository for interaction datasets. *Nucleic Acids Res*, 34:535–539, 2006.
64. R. Steuer, J. Kurths, C.O. Daub, J. Weise, and J. Selbig. The mutual information: Detecting and evaluating dependencies between variables. *Bioinformatics*, S18:231–240, 2002.
65. M.C. Teixeira, P. Monteiro, P. Jain, S. Tenreiro, A.R. Fernandes, N.P. Mira, M. Alenquer, A.T. Freitas, A.L. Oliveira, and I. Sa-Correia. The YEASTRACT database: A tool for the analysis of transcription regulatory associations in Saccharomyces cerevisiae. *Nucleic Acids Res*, 34:446–451, 2006.

66. L. Torgo. Functional models for regression tree leaves. *Proceedings of the 14th International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, CA, 1997, pp. 385–393.

67. R.S. Wang, X.S. Zhang, and L. Chen. Inferring transcriptional interactions and regulator activities from experimental data. *Mol Cells*, 24:307–15, 2007.

68. F.C. Wimberly, T. Heiman, J. Ramsey, and C. Glymour. Experiments on the accuracy of algorithms for inferring the structure of genetic regulatory networks from microarray expression levels. *Proceedings of the IJCAI 2003 Bioinformatics Workshop*, 2003.

69. X. Zhang and W. Wang. An efficient algorithm for mining coherent patterns from heterogeneous microarrays. *19th International Conference on Scientific and Statistical Database Management, 2007, SSBDM '07*, 2007, p. 32.

42

# APPROACHES TO CONSTRUCTION AND ANALYSIS OF MICRORNA-MEDIATED NETWORKS

Ilana Lichtenstein, Albert Zomaya, Jennifer Gamble, and Mathew Vadas

## 42.1 INTRODUCTION

### 42.1.1 miRNA-mediated Genetic Regulatory Networks

A genetic regulatory network (GRN) can be thought of as a network made up of molecular species and their interactions, which together control the levels of gene expression and end products in that network. The research effort to create formal models of GRNs and develop tools to infer such models by integrating high-throughput data and other data sources is a well-established field in systems biology.

Consistent with traditional dogma in theoretical biology, models of GRNs have to date focused on identifying protein-mediated regulation of gene transcription by transcription factors (TFs). However, modern theoretical biology postulates that complexity in higher organism GRNs may be mediated via alternative mechanisms involving sequence-specific binding of noncoding RNA (nc-RNA) molecules to a range of targets [49]. nc-RNAs have been implicated in regulation of gene expression programs at many levels, including chromatin modification, transcription,

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

alternative splicing, RNA modification, RNA editing, mRNA translation, RNA stability, and cellular signal transduction and trafficking pathways.

Computational methods and tools must be adapted to provide accurate and meaningful models of GRNs in light of changing paradigms in biology, with increased focus on nc-RNAs. This chapter reviews the application of computational methods to modeling GRNs, or parts of GRNs, where microRNAs (miRNAs) function as important regulators of network behavior. We have focused on miRNA-mediated regulation because miRNAs are a relatively well-studied class of ncRNAs, and because they operate on a large scale, targeting a high number of genes in the genome, and thereby regulating many important biological processes.

microRNAs (miRNAs) are a class of small regulatory nc-RNAS that are $\sim$22 nt long. miRNAs are transcribed in the nucleus as pri-miRNAs after which the transcript is processed into a 60–70-nt hairpin structure (pre-miRNA), with the mature miRNA located on one or both arms of the hairpin stem. The hairpin structure is transported to the cytoplasm where it is further processed and the mature miRNA transcript is released. See [7] for further details of the miRNA biogenesis pathway.

miRNAs are thought to play a role in developmental timing, cell proliferation, apoptosis, metabolism, and morphogenesis [1, 2] and to play a role in many cancers [47]. miRNAs are believed to regulate cellular programs at multiple levels; however, the most well-studied level is translational repression, which is mediated through miRNA antisense complementary binding to a target region in the 3’UTR of a mRNA target. It is believed that in plants miRNA perfectly binds mRNA substrates, which initiates degradation of the transcript [20]. In animals, it is thought that miRNAs sometimes bind weakly with imperfect complementarity to their target, which leads to translational repression at the site of the ribosome [2, 7, 13]. However, other studies have demonstrated that some miRNAs act at the mRNA transcript level in both plants and animals through degradation or deadenylation [4, 45].

There are now 8619 known miRNA hairpin loci according to the miRBase Sequence Database [25, 26, 27] (miRBase, v.12.0, September 2008). At least 10–40% of genes are regulated by miRNAs in vertebrates, insects, and nematodes (reviewed in [28]), with many target transcripts thought to be targeted by coexpressed miRNAs [39].

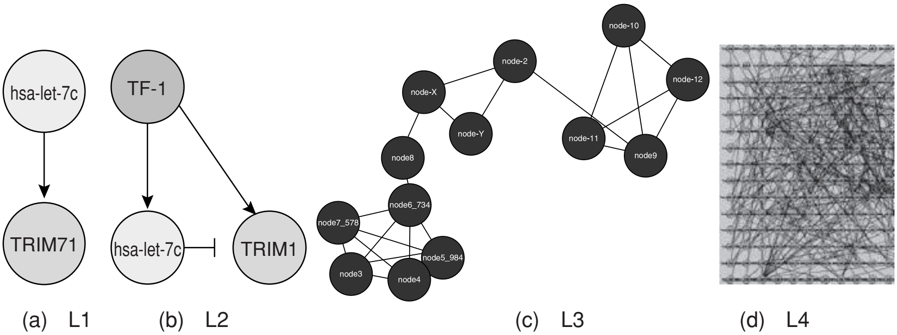
miRNAs therefore affect cell behavior. Such effects can be seen qualitatively and quantitatively with mathematical models of network dynamics. However, such models rely on proper understanding of the underlying network structure and properties. Thus, there is much motivation to develop tools that can aid in computational inference of the structure and behavior of components that make up miRNA-mediated networks.

Conceptually, one might expect GRNs that incorporate miRNA-mediated mechanisms of control to share similar types of network components, interactions, and architectural properties with TF-mediated networks because TF-mediated and miRNA-mediated control of gene expression share many common functional characteristics.

Both are *trans*-acting molecules that activate or repress genes via hardwired *cis*-regulatory binding sites, and both use coordinated and cooperative target binding to fine-tune their instructions contextually [31]. We might therefore expect that similar formalisms and reconstruction methods may be applied and adapted to describe, infer, and analyze these networks.

### 42.1.2 The Four Levels of Regulation in GRNs

According to Babu *et al.* [3] transcription regulatory networks (TRNs) may be considered to have four layers of regulation. TRNs are a subset of GRNs that focus on transcriptional control and, thus, are a good representative of GRNs for the purpose of categorizing layers of regulation. At the lowest level, individual regulatory molecules and their targets interact directly according to biochemical reaction laws. At the next level up, local circuits are formed by interconnecting individual interactions. Such circuits can perform operations on input signals, leading to a change in form or quantity of one or more output components. Circuits can be considered to be patterns defined by graphical properties such as number of nodes and edges, and edge directionality. Where these patterns are found to be overrepresented in a TRN or GRN against the expected number of such patterns in randomly generated networks, they are termed network "motifs." At the next level up, transcription modules are identified as sets of genes that show some measure of similarity to one another within each module, and some measure of independence from genes in other modules, where the level of similarity or difference is determined by properties of the genes such as promoter sequences or expression profiles (*i.e.*, clustering). At the highest level, the TRN comprises an integrated network comprising the interaction of the individual modules [3]. Figure 42.1 describes the four levels of detail with which we can conceive a GRN.



(a) L1 (b) L2 (c) L3 (d) L4

**Figure 42.1** GRNs can be conceptualized at four levels of detail. (a) Level 1 displays direct interactions between genes. (b) Level 2 displays circuits made up of level 1 interactions. (c) Level 3 displays modules of genes comprising many circuits, where each module displays semi-independent behavior. (d) Level 4 conceptualization of the network contains all possible interactions, circuits, and modules.

### 42.1.3 Overview of Sections

Section 42.2 reviews efforts to identify the basic components and interactions—miRNA genes, their transcription properties and regulators, and their mRNA targets—which make up miRNA-mediated networks. Computational research into predicting these low-level components and interactions is extensive, and the section does not provide an exhaustive discussion of such research but provides a high-level overview, referring the reader to more detailed reviews where appropriate. Section 42.3 reviews methods that identify network subgraphs at the circuit level (level 2) and the module level (level 3). Section 42.3.1 describes studies that identify individual circuits by integrating miRNA-mediated basic interactions with other validated interactions (*i.e.*, *forward engineering* of the network). These methods are straightforward, and the subsection only exists to contrast against reverse-engineering approaches. Section 42.3.2 reviews efforts to identify modules comprising genes (including miRNAs in some instances) where there is evidence that those genes within each module share similar properties. In the case of miR-mediated network, the concept of a "module" is somewhat different to a transcriptional module. Properties that define miRNA-mediated modules include (dis)expression levels of the miRNAs and mRNAs under various conditions, and the gene or miRNA sequences that may bind common regulators (TFs or miRNAs). These methods often involve *reverse-engineering* modules from high-throughput data using statistical or machine learning techniques, where constraints are imposed by stored prior biological knowledge (*e.g.*, miRNA-target knowledge and conservation requirements).

Section 42.4 reviews studies into global and local topological properties of miRNA-containing networks and efforts to map network motifs to suggested miRNA cellular functions.

## 42.2 FUNDAMENTAL COMPONENT INTERACTION RESEARCH: PREDICTING MIRNA GENES, REGULATORS, AND TARGETS

There are few miRNA-spanning networks that are experimentally mapped out and well understood. Instead, the process of reconstructing miRNA-containing networks has mostly been done in a somewhat piecemeal fashion, with each research study leading to the addition of one likely connection (molecular interaction) that may form part of many different miRNA-containing networks. Computational prediction of such interactions involves methods that screen genomes searching for the presence of statistically improbable features that are characteristic of the interaction type. For example, to find potential miRNA regulators, features searched for include transcription factor binding sites (TFBS) in the promoter or enhancer regions of genes, or for potential miRNA-target interactions, complementarity binding seeds in 3'UTR region of target mRNA transcripts. Often, multiple genomes are screened to assess feature conservation across species. Predicted and validated interactions are compiled into online catalogues that can be searched online or downloaded (see discussion below).

### 42.2.1 Prediction of Novel miRNA Genes

Chaudhuri and Chatterjee provide a comprehensive survey of computational methods developed to predict novel miRNA genes, including plant and viral miRNAs [14]. Many methods involve searching candidate sequences in genomes for features that suggest the presence of a pre-miRNA hairpin. See, also, Leung *et al.* in [43]. The features searched for include free energy of the putative miRNA hairpin (indicating thermodynamic stability), the number of paired bases in the stem (indicating structural accessibility for enzymes to cleave mature miRNA), and the sizes of the loops and bulges indicating the ability of mature miRNA to be loaded into the RISC complex in which it travels to the target [62].

Xie *et al.* screened the promoter regions and 3’UTR regions of protein coding genes across human, mouse, rat, and dog genomes in a multipart study to identify novel putative miRNA genes by searching for potential binding sites in 3’UTRs of target genes [84]. Their study involved searching for highly conserved (HC) sequence motifs in 3’UTRs, which are sequences of a certain length that have a significantly high ratio of conserved occurrences across four species against a set of random background sequences sampled from the same region in the human genome. When searching for HC motifs in 3’UTRs, they found 106 HC motifs in the 3’UTRs, identified a directional bias in the HC motifs with respect to DNA strand, and found a peak in the number of HC motifs of length 8. Of note, no similar peak was found for motifs during similar studies in the promoter region of genes. An analysis of a selection HC 8-mer motifs were clustered into 72 HC similar motifs that matched 46% of the full set of 3’UTR motifs. They then screened for new miRNA genes by searching the four aligned genomes for sequences complementary to the 72 3’UTR 8-mers. Sequences flanking the conserved site were then evaluated with the RNAfold program to identify those with stable stem-loop structures. At the time of the study, there were 222 known miRNA genes, 113 of which were found in the HC sequences. The study further predicted 129 new miRNA genes from these sequences. Of a representative set of 12 predicted genes, 6 were validated to be genes expressed in adult tissues. The authors conclude approximately 45% of the HC motifs are related to miRNAs.

Studies such as Xie *et al.* can assist in the identification of novel microRNAs; however, as the authors discuss, many (around half) of the known miRNAs were not found to bind to highly conserved motifs. This suggested to the authors that many miRNAs may bind regions outside the 3’UTR, or may be partially complementary to their targets. Many miRNAs may evolve rapidly and are thus not highly conserved, nor do they possess highly conserved targets. Indeed, some studies predict that there may be up to 800 human miRNAs [10] (or up to around 1000 miRNAs conserved across vertebrates [11]). If indeed variation across genomes is highest at the level of nc-RNAs, then using conservation as a screening criteria may limit the ability to detect recently evolved, species-specific miRNAs. Thus, when building miRNA networks from the bottom up (*i.e.*, linking known components), it is important to be aware that we do not yet have a complete inventory of miRNAs, including the many species-specific miRNAs.

The reader is referred to Chaudhuri and Chatterjee for further studies.

### 42.2.2 Prediction of miRNA Targets

An even larger effort has focused on identifying specific miRNA target sites in mRNAs. Many researchers have developed mRNA target prediction algorithms, and there exist several online databases that predict binding sites and target genes of individual miRNAs. Many of these websites contain precompiled lists (catalogues) of miRNA target predictions, such as PicTar [39], TargetScan [41], and TargetScanS [42]. Good reviews of target prediction algorithms exist elsewhere [14, 28, 60].

These methods have varying levels of accuracy, and in practice, studies often combine results from different online databases (which use different sets of the above criteria in their search algorithms) to reduce noise in the overall prediction. Sethupathy *et al.* [60] evaluate the performance of five well-known target prediction algorithms against experimentally validated targets from TarBase [61]. The study also evaluates the success of predictions based on unions and intersections of the different methods.

Sethupathy *et al.*'s study and others reveal that there is still great diversity among the target prediction algorithms. For example, in the study by Sethupathy *et al.*, the intersection of PicTar and TargetScanS in detecting conserved miRNA-target gene interactions from an unbiased dataset achieved just slightly less sensitivity (56.7%) than either program individually (64.3% for TargetScanS and 63.2% for PicTar). However, adding miRanda to the intersection dropped the sensitivity to nearly 40%, and adding either DIANA-microT or TargetScan to the intersection reduced to around 10%. Further research has raised concerns regarding reliance on unions and intersections of target prediction tools [55]. Thus, use of target prediction algorithms need to be treated with caution.

Up to 90% of target genes have been validated with assays in some target prediction algorithms [54]. TarBase provides a list of experimentally verified target predictions, including annotations. The TargetCombo website (http://www.diana.pcbi.upenn.edu/cgi-bin/TaretCombo.cgi) provides precompiled lists of results from selected combinations of programs that are discussed.

### 42.2.3 Prediction of miRNA Transcript Elements and Transcriptional Regulation

Efforts to understand direct miRNA interactions can be improved by incorporating information on the miRNA transcriptome. Such studies assist in locating and analyzing promoter and enhancer elements of miRNA genes, which provide information on which TFs regulate the gene, and under which conditions. Large-scale screening of genomic regions upstream and downstream of putative or known miRNA genes have shed light on the transcriptional apparatus of miRNAs.

Little is known about transcripts of independently transcribed miRNAs, including intergenic miRNAs, miRNAs that are intronic but antisense to their host gene, or intronic miRNAs that may lie in sense to their host gene yet possess autonomous transcription machinery. On the other hand, transcriptional apparatus of intragenic miRNAs that are thought to be transcribed together with their host gene may be better understood where transcription of the host gene has been explored.

Ohler *et al.* studied upstream and downstream regions of 88 independent and 13 co-transcribed miRNA stem-loop foldbacks in *Caenorhabditis elegans* [52]. The search was motivated by the desire to improve the performance of their miRNA gene finding algorithm, miRScan by identifying additional sequence features of miRNA genes with which to train the model. The authors first searched for conserved upstream or downstream regions of candidate genes in order to search for features within these regions. A peak was found in conservation about 200 bp upstream of the conserved stem-loop foldbacks. The authors then searched from motifs in these regions, using both an enumerative word-based algorithm, the ST algorithm, based on an approached described by Sinha and Tompa [66], and an alignment-based motif finding tool, MEME [5]. Both algorithms identified one highly significantly motif that was overrepresented in the upstream region of independently transcribed miRNAs, which was not found in co-transcribed miRNAs or other genes. The authors also used the algorithms to search for motifs in the upstream regions of miRNAs in mammals and *Drosophila melanogaster*.

Sethupathy *et al.* investigated the location of promoter regions of intergenic miRNAs in human and mouse by assembling contiguous ESTs downloaded from dbEST, which overlap the intergenic miRNA precursor sequences [59]. They then searched for motifs in the upstream regions of the putative primary transcripts and scanned significant motifs for vertebrate TFBS. Several sites were found to recruit TFs known to be involved in development. Furthermore, the authors investigated whether miRNAs expressed in a tissue-specific manner possess any specific motifs or TFBS not shared by other miRNAs. After clustering miRNAs according to expression profiles based on northern blots and microarrays, they selected the brain cluster for further investigation and searched for upstream motifs and TFBS of miRNA genes within the cluster. However, due to limited ESTs in the dbEST database, none of the miRNAs in the brain cluster had EST support so no putative primary transcript was created, and thus, the study was limited to searching upstream of the precursor sequence.

Gu *et al.* studied the expression profile of mammalian intergenic miRNAs by matching expressed sequence tags (ESTs) downloaded from dbEST to miRNA flanking sequences [29]. The authors argue that this provides a cheaper and faster alternative to microarrays and cloning to study microRNA expression patterns. RT-PCR was performed to confirm the expression of miRNAs matched to the predicted ESTs. Information on miRNA expression behavior could then be gleaned from the annotation of the matched EST.

Similar studies exist that have examined the promoter regions of plants [50, 75]. Xie *et al.* identified the promoters of 52 *Arabidopsis thaliana* microRNA genes and showed that most of them have TATA-boxes in their core promoters [75].

Zhou *et al.* studied and characterized promoter regions of intergenic miRNAs on a genome-wide scale in four species, including human, nematode, and plant species [78]. The authors acknowledged the difficulty of building a prediction model based on learning key *cis*-regulatory elements due to the variability in the presence of elements found in promoter regions of known genes, and further due to the relatively unchartered area of miRNA promoter analysis. In the first part of the study, they built a classifier that could discriminate between input sequences that were Pol II promoters, Pol III promoters, and random intergenic sequences. For each species, two classifiers were used, one based on a decision tree model, whereas the other employed a support vector machine.

The features used to train the model comprised k-mers that were found to be overrepresented in known Pol II promoters, Pol III promoters, and random genes. The machines were trained using a feature vector for each promoter that specified the number of statistically overrepresented motifs present in that gene for each feature. The authors used 10-fold cross-validation to assess the accuracy of the classifiers. They further developed a novel promoter prediction algorithm, CoVote, which they used to predict novel promoters for the intergenic miRNAs in the four species. The model is trained using motifs obtained from upstream regions of known Pol II genes. A decision tree (DT) model is used to classify segments of an input sequence in a core promoter class (Pol II or random), and the path to the leaf node in the DT specifies the set of motifs present in the segment. Leaves of the tree are weighted according to the number of segments that have followed the same sequence motif path to arrive there and, thus, reveals common votes for this path by other segments, hence, the name *common query voting* (coVote). Each miRNA gene is then considered to have a core promoter region as a concatenation of the segments found in its upstream region whose leaf node is scored above a cutoff score.

Their study confirmed prior studies suggesting that many microRNAs share common elements in their promoters to protein-coding genes and are therefore likely to be transcribed by RNA polymerase II (Pol II). They identified many *cis*-regulatory elements common to intergenic miRNA promoter regions in all four species, and some that are species specific.

Lee *et al.* developed a method that screened the upstream regions of known human miRNA genes to identify *cis*-regulatory motifs (CRMs) *de novo* [40]. The background sequences were taken from the genome of interest to ensure they represented the true background distribution of sequences given high representation of some sequences due to effects of evolution of the genome (*e.g.*, genomic expansion). CRMs also could be used to identify miRNAs that are likely to be regulated by the CRMs. The CRMs identified could be used to predict TFBS *de novo*. Top-scoring CRMs were compared against a set of motifs known to be binding sites of human transcription factors in the TRANSFAC database [48]. A vast number of CRMs found upstream of the miRNA genes also were located within 500 nt of protein-coding genes, suggesting similar factors control transcription of genes and miRNAs.

The authors identified miRNAs that they predicted were combinatorially controlled by TFs due to the existence of multiple CRMs in the upstream region. The study then focused on 48 miRNA genes that were predicted to be regulated by the top 50 hexamer CRMs. Using information on TF-CRM and CRM-miRNA interactions, the authors predicted that several TFs regulated all 48 miRNAs, and thus, the TFs were considered to be master regulators of miRNA expression. Several miRNAs were predicted to be combinatorially regulated by several CRMs (*e.g.*, miR-132 was predicted to be regulated by 24 CRMs), suggesting that miRNA expression is finely controlled, and supporting the notion that faults in a complex transcriptional coordination program may underlie miRNA misexpression in cancer.

To investigate properties of the intergenic miRNA transcript more generally, Saini *et al.* analyzed the genomic location of several transcription elements relative to 474 human pre-miRNAs in miRBase [56]. The features analysed were transcription start sites, CpG islands, ESTs, TF-binding sites, expression ditags, and poly(A) signal predictions. Combining results of the analysis of each of these features, the authors can delineate several 5′ and 3′ boundaries of a transcript region for each miRNA or for cluster of miRNAs. Of interest, it was found that the transcripts of a significant fraction of intergenic miRNAs were 3–4 kb in length, with a small fraction of longer transcripts up to 6 kb.

The location of the TSS for intergenic regions were peaked within 2 kb upstream of pre-miRNA start, with a smaller peak at 9–10 kb upstream. This was in contrast to intronic pre-miRNAs whose TSS peaked at around 2–6 kb upstream. Many miRNAs clustered within a distance of 10 kb were found to share a similar TSS, and the authors concluded these miRNAs are co-transcribed. A significant proportion (40%) of CpG islands were found to be overlapping with predicted TSS sites within 4 kb upstream. Poly(A) signals that predict 3′ boundaries peaked at 2 kb downstream of the pre-miRNA. Results obtained from the large-scale genomic study were validated using miRNA expression data via EST matches and gene identification signature-paired-end digtags (GIS-PET). The authors produced a canonical structure of the mammalian miRNA primary transcript based on their findings.

In a follow-up study, Saini *et al.* extended their study into the transcript boundaries of pri-miRNA to regions conserved across species in order to identify "consensus features" that are features conserved across pri-miRNAs in multiple sequences [57].

Fujita *et al.* adopted an approach that they argued improved on the work of Lee and Zhao by searching for promoter regions by taking into account the structure of the pre-miRNA and the transcription unit. They identified sequences that showed conserved miRNA hairpins and screened regions 100 kb upstream of these conserved gene regions to identify putatitive promoter regions (miPPR) that were conserved across species, and contained at least one core promoter element required for transcription by RNA polymerase II (TATA, CCAAT, and GC box) [24]. Where more than one promoter region satisfying these requirements was identified for an miRNA, the region in closest proximity to the miRNA gene was selected as the miPRR. Biochemical analysis of the miRNA transcript was conducted using primer extension and Northern blots to confirm that the miRNA transcript was indeed driven by elements in the miPRR.

As with computational prediction of novel miRNA genes, the success of identifying novel binding sites using genomic screening methods is limited by the experimental design of the study. Screening for motifs that lie in an upstream region of a certain distance from the transcription start site will not detect CRMs that may lie several thousand nucleotides away from miRNA sequences, do not lie within the same strand of DNA, or are not in a contiguous stretch of DNA. Furthermore, as algorithms rely on either consensus of motifs lying upstream of the miRNA gene within a species, or conservation across species, promoter regions that do not possess sequence elements that are significantly overrepresented according to the model may fail to be detected.

## 42.3 IDENTIFYING MIRNA-MEDIATED NETWORKS

### 42.3.1 Forward Engineering—Construction of Multinode Components in miRNA-mediated Networks Using Paired Interaction Information

The role of miRNAs in regulatory circuits in development systems is the subject of interest. Many studies use genetic analysis studies to infer the presence of individual interactions and multinode components involving miRNA regulation. Such approaches may be considered to be forward-engineering or bottom-up approaches to network construction.

In studies aimed at identifying the molecular architecture responsible for terminal differentiated states of *C. elegans* taste receptor neurons, “ASE left” (ASEL) and “ASE right” (ASER). Johnston *et al.* used a series of mutation experiments and gene expression profile analysis in which ASE symmetry is disrupted to infer causative relationships between genes. The output architecture placed two miRNAs (miR-273 and lsy-6) in a double-negative feedback loop together with die-1, the output regulator of downstream effector genes [35]. The network architecture is believed to govern bistability and irreversibility of ASE neuron specification in response to an unidentified input signal. Other examples where a forward engineering approaches are used to identify architecture of miRNA-mediated circuits. See also [44, 53].

### 42.3.2 Reverse Engineering—Inference of MicroRNA Modules Using Top-Down Approaches

The fields of mathematics and computational science have provided important methods and tools that aid in inferring the structure of GRNs from large-scale data sources such as microarrays, most often through the use of statistically driven algorithms. Such methods can thus be used to identify modules within expression data (level 3 regulation), or an attempt may be made to reconstruct relationships across the entire network (level 4 regulation). A wide variety of methods exist for reconstructing relationships across the network from high-throughput expression data, including methods that specify genetic networks as Boolean networks [65], Bayesian networks [23, 51], and linear models [12, 18]. The reader is referred to more detailed reviews in [19] and [38]. Such methods and tools have thus played an important role in the effort to map out genetic processes that regulate cell programs. Methods used to identify regulatory modules *de novo* in transcription networks that share common transcriptional profiles have been explored in [9] and [58].

Recently, the concept of miRNA modules (MRMs) have been developed, which are sets of genes comprising miRNAs and mRNAs, where the genes within a module possess some measure of similarity determined by a set of properties, where that similarity is less strong between genes in different modules. In the first subsection, we consider gene clusters that are similar to transcription modules, in that they comprise the targets of a single miRNA. Then we consider research into the recently developed MRM.

***42.3.2.1 Inference of miRNA-associated Gene Clusters.*** Several studies have been conducted that have aimed to infer an association between individual miRNAs and tissue-specific gene signatures. Sood *et al.* examined the expression of mRNA genes containing a nucleus or binding site for a particular miRNA in tissues where the miRNA was overexpressed. The removal of PicTar targets (genes containing multiple binding sites) dampened the degree of downregulation of the predicted targets, suggesting that the existence of multiple binding sites for the miRNA was an important factor in miRNA downregulation [67].

Zilberstein *et al.* developed an algorithm to infer conditions in which individual microRNAs are active [77]. The methodology involved creating two matrices: one that specifies the predicted targets of miRNAs, and another that contains the expression data for mRNAs over a range of conditions. A function produced a score for each miRNA based on the two matrices by comparing the number of its predicted targets that were downregulated in a condition against the number of its nontargets that were downregulated in the condition. A statistical analysis of the scores allowed the authors to infer which miNAs were implicated in each condition. The experiment was performed on 98 miRNAs under 380 conditions in *A. thaliana*. Several tissue-specific miRNAs were determined.

Huang *et al.* developed a program GenMir [33], and its improved version, GenMir++ [34], which use variational Bayesian learning methods to detect whether putative targets predicted by target prediction algorithms are functional. The authors developed a probabilistic graphical model to represent the probability that a predicted mRNA $x_i$ was downregulated by its predicted regulator, microRNA $m_j$, using an unobserved binary random variable $s_{ij}$, where $s_{ij} = 1$ for a functional prediction and $s_{ij} = 0$ otherwise. The inference task involves finding the posterior probability for each value $s_{ij}$ in the set $S$, that is, the posterior probability that the target prediction is valid conditioned on the expression data. The advantage of GenMir over Zilberstein's methodology is that it takes into account coordinated miRNA regulation by analyzing miRNA expression data as well as mRNA expression data and sequence data. GenMiR++ significantly improves the performance of GenMir by including additional parameters to take into account tissue scaling effects for the miRNA and mRNA arrays.

Cheng *et al.* studied the activity of 211 individual human miRNAs on genes, by analyzing 12 expression change profiles of their predicted targets from six miRNA transfection experiments taken at 12 hours and 24 hours [15]. The expression data from each of the 12 profiles were integrated with information on predicted binding ability using the miRanda algorithm. They use an activity change (AC) score to infer the relative activity of each miRNA across two conditions (transfection and reference). The algorithm ranks genes from highest to lowest expression change values in the array. A prescore is calculated as a relative measure of the binding affinity of the transfected miRNA among highly or lowly expressed genes in the ranked list, against a control list. In other words, if high binding affinity genes are enriched at the top or bottom of the list, this indicates the miRNA is itself downregulated, or upregulated, respectively. The score is then normalized, and its statistical significance is determined. The authors found that in the miR-1 and miR-24 transfection experiments, it could be inferred that some other miRNAs were also downregulated. The authors postulated that this could be caused by either one miRNA (*e.g.*, the transfected miRNA) affecting other miRNAs, or alternatively, that the introduction of exogenous miRNA affected the processing and maturation of endogenous miRNAs.

***42.3.2.2 Inference of Multiple Interaction Modules.*** Yoon and De Micheli developed the concept of a microRNA module (MRM), which can be defined as a set of miRNAs and their target genes believed to cooperate in post-transcriptional regulation [76]. Figure 42.2 is an example of an MRM represented as a relation graph.

Yoon and De Micheli use a five-step process to search for MRMs among a set of input miRNAs and a set of input genes using target prediction data. In developing their algorithm for finding MRMs, the authors were motivated by the theory that key features of miRNA control are a multiplicity of targets per miRNA and a multiplicity of input regulation for each target. They assume MRMs favor many-many relations and that the binding strength of each miRNA on a target should be similar. The first step in the method involves identifying possible miRNA–mRNA duplexes among the sets using a local alignment score and free energy of the miRNA–mRNA duplex. In the next step, a weighted bipartite graph (the relation graph) is constructed between the miRNAs and the predicted targets, where the weight of the edge between each miRNA and its target is determined using principal component analysis.

The third step involves finding maximal bicliques in the relation graph. Bicliques are complete subgraphs in a bipartite graph, and maximal bicliques are bicliques that do not form a proper subgraph of another biclique. The algorithm searches for seeds that are a set of miRNAs for each target $t$ in the relation graph, where the miRNAs in the seed have edges incident on $t$ with a similar weight. The fourth step uses an algorithm to search for MRMs by constructing a trie where seeds are stored in the nodes of the tree, and then merging nodes to find candidate MRMs from those nodes with the highest number of target genes. In the final step, statistically significant MRMs are selected from candidate MRMs. A Poisson random variable is assumed for the number of MRMs in the relation graph. The result was tested on human miRNA and gene sequences and detected 431 MRMs with on average 3.58 miRNAs

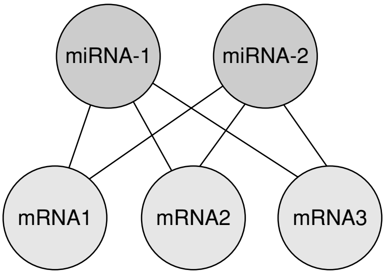


**Figure 42.2** An example of an MRM represented as a relation graph.

and 6.74 target genes, and ontological analysis was performed on one high-scoring module.

Joung *et al.* use a co-evolutionary, population-based probabilistic learning algorithm that uses both target prediction and expression data to identify MRMs [36]. The method searches through possible miRNA–mRNA modules to find optimal modules. The algorithm initializes two population vectors $X$ and $Y$ with individuals $x_i$ and $y_i$, where each individual represents a binary string reflecting miRNAs or target mRNAs with values set to either on or off, respectively. Each individual can therefore be thought of as a set of unique miRNAs (M) in $x_i$ or mRNAs (T) in $y_i$. A fitness function is used to evaluate each individual $x_i$ $(x_j)$ against another individual $x - j$ $(x - i)$ from the other population. This score is based on a balanced aggregate of the mean target prediction score of the miRNAs in M against targets in T, and the expression coherence of the subsets M and T. For each individual $x_i$ $(x_j)$, the individual in the other population $x - j$ $(x - i)$ producing the highest score is used. The most fit individuals are thus selected. A set of probability vectors specifies the probability that each miRNA (or mRNA) will be included in the next generation of individuals. After each evaluation round, the probability vectors are updated based on the frequency of occurrence of the miRNA (or mRNA) in the most fit individuals selected in the previous generation. This step is repeated for a specified number of generations, and optimal modules are selected. The authors discovered modules among 3982 miRNA–mRNA target pairs in 89 cancer samples. They investigated two statistically significant modules; the first one was found to contain miRNAs that were tumor-suppressors, whereas the second module was also found to also be cancer-related.

Tran *et al.* improve on Joung's method by using CN2-SD, a type of separate-and-conquer, rule-based induction method to find MRMs from PicTar target prediction data, and miRNA and mRNA expression data [70]. The rules are learned from a set of examples that comprise a table for each mRNA (subject mRNA) in the dataset. The table contains a row for each other mRNA in the dataset, and each column specifies whether the mRNA is a predicted target of each miRNA in the dataset, and whether the gene expression of the mRNA and the subject mRNA are correlated (*similar* using Pearson's correlation coeffeicient). A CN2-SD rule induction system learns a set of rules of the form *Condition* → *Class*. In this instance, the condition is a conjunction of attribute value pairs, here values of miRs, and the class is the number of positive (subject mRNA is correlated or *similar*) or negative (subject mRNA is not correlated or *dissimilar*) training instances covered by the rule.

Noninteresting rules are filtered out, so that rules that remain have a high expression correlation between genes and miRNAs within each rule. MRMs are selected from each rule by selecting examples with *similarity* class covered by the rule.

Liu *et al.* [46] use association rule mining to find functional MRMs (FMRMs), which are MRMs associated with a particular condition. They first find sets of maximal bicliques from miRNA–mRNA target pairs predicted from miRBase Targets version 5.0. The miRNA and mRNA expression profiles of 12 prostate cancer samples (6 cancer and 6 normal) are downloaded and discretized. Frequent itemsets are subsets of each maximal biclique (*e.g.*, miR-1, miR-2, mRNA-1, mRNA-2 and mRNA-3), where the miRNAs and the mRNAs show opposite expression patterns (*e.g.*, *miR-1* ↑, *miR-2* ↑, *mRNA-1* ↓, and *mRNA-2* ↓). They then used frequent itemset mining to discover whether a frequent itemset associates with a condition. This leads to a set of relations comprising the itemset and its association (*e.g.*, cancer), which are the FMRMs. Redundancies (*i.e.*, where miRNA–mRNA patterns are shared between more than one itemset) are removed by merging the itemsets to produce an FMRM for the condition.

Joung *et al.* use an author-topic model that is family of a probabilistic graphical model to predict miRNA modules (which they term miRMs), which takes into account gene expression data, without requiring miRNA expression data [37]. They also use transcription factor binding elements and miRNA promoter analysis to determine likely regulators for the miRMs.

***42.3.2.3 Discussion.*** The use of expression data as evidence in algorithms that learn MRMs suffers from several limitations. First, gene expression data reveal levels of mRNA transcripts usually as a proxy for protein levels; however, in animals it is known that sometimes miRNAs exert their influence through translational repression. Therefore, the repressive effect or miRNA regulation may not be detected at the level of mRNA expression. Thus, using anticorrelation of expression between miRNA and mRNA as input to a module finding algorithm may ignore relationships where the miRNA and the mRNA remain correlated, but the protein level of the target gene is affected. Proteomic analysis may provide valuable data in order to overcome this limitation [22].

Second, in some situations, the miRNA is not the primary regulator of its target, and its role is to maintain the target at steady-state levels. In such a situation, the target transcript or protein product abundance is only moderately reduced. This means the response variable is "tuned" rather than "switched-off." Again, module finding algorithms that search for anticorrelated behavior between miR and its predicted target may miss more subtle modulations. Third, there may be some situations where only a small fold change in the target mRNA is required to mediate a functional response that causes the cell to switch to a different state; yet module finding, algorithms may ignore such targets because the fold change is below a predetermined cutoff.

Furthermore, in some cases, a phenotypic effect is mediated from a "target battery" comprising all repressed targets; yet in other cases, only several targets are actually required to mediate the phenotypic effect by acting as a genetic switch. Thus, we may consider a module to comprise a set of genes showing widespread repression, even though only some of these are required for the phenotypic response of interest. By examining the module in question (such as with gene ontology analysis tools), we will not be able to distinguish the roles of different genes in the module.

Despite this limitation, module identification is an important step in identifying subsets of genes and the miRs that regulate them, which may be involved in mediating a functional response. Gene ontology studies may reveal functional categories enriched in the modules.

## 42.4 GLOBAL AND LOCAL ARCHITECTURE ANALYSIS IN MIRNA-CONTAINING NETWORKS

Analysis of global topological properties of transcription regulatory networks (TRNs) is well established. Many cellular networks display a scale-free architecture (see discussion in [6]). For example, TRNs in *Saccharomyces cerevisiae* and *Escherichia coli* have been found to have a scale-free architecture with respect to the *out-degree* of transcription factor proteins, meaning that most TFs regulate very few partners, but some TFs regulate many partners. However, TRNs have been found to display a restricted exponential distribution with respect to *in-degree*, meaning that most genes are regulated by one, two, or three regulators [3, 30]. Similarly, there has been much research into representation of local patterns within GRNs or TRNs. Such motifs are generally three-node and four-node components capable of signal processing [64]. Recently, parallel research into the global and local properties of miRNA regulation at the post-transcriptional level has commenced.

Post-transcriptional networks can be constructed from prediction or experimentally validated interaction data for the purpose of characterizing global and local properties. In the case of TF-mediated networks, studies into local motif properties were sometimes conducted on networks that were already well understood; network analysis was therefore a separate task to network identification. In the case of miRNAs, little is known about complete networks containing multinode components because experimental validation of indiviual miRNA genes and target binding interactions is still relatively new. Thus, the process of individual network construction using forward- and reverse-engineering approaches is often done in parallel with studies into global and local network properties.

### 42.4.1 Global Architecture Properties of miRNA-mediated Post-transcriptional Networks

***42.4.1.1 Scale-free Properties of miRNA-containing Networks.*** Shalgi *et al.* studied global and local properties of miRNA-mediated networks in an attempt to characterize properties of coordinated regulation by TF-miRNA, or by miRNA–miRNA pairs. The authors studied predicted interactions of a set of miRNA and target genes downloaded from the Pictar [39] (178 miRNAs and 9152 human RefSeq genes) and TargetScan [41, 42] (138 miRNAs and 8672 human RefSeq genes). Both of these databases contain predicted binding site information for evolutionarily conserved miRNA-target pairs. The authors identified a long-tail (*i.e.*, scale free) topology for the number of different miRNAs regulating each gene (*in-degree*). They named genes with a high in-degree as gene “target hubs” to indicate their analogy to transcription factor gene target hubs demonstrated elsewhere. These genes were those that displayed a statistically high number of miRNA binding sites or, alternatively, a high density of binding sites against random sequences. An ontological analysis revealed target hub genes were enriched for TFs that play important roles as regulators in processes such as development and transcription. This finding extended the scope of the theory applied to TF-mediated yeast networks, that genes that are significant regulators are themselves heavily regulated.

Martinez *et al.* perform a similar study on global properties of both TF − > miRNA and miRNA − > TF regulatory networks in *C. elegans*. In the first scenario, they use the yeast one-hybrid (Y1H) method that allows detection of TFs that can bind to a set of promoters of interest. They perform the study on the *C. elegans* genome, which has 115 known miRNA genes in miRBase V4.0 and WormBase WS130, and 940 predicted TFs according to previous studies, to obtain 347 high-confidence interactions between 63 miRNA promoters and 116 proteins. An examination of the high-confidence TF− >miRNA interactions revealed a power law for the out-degree distribution of the TFs ($R^2 = 0.82$) and an exponential law for the in-degree distribution ($R^2 = 0.84$) of the miRNA promoters. This finding for miRNA transcription regulatory networks is therefore similar to findings relating to global regulation in protein-coding gene networks.

They authors also examined the interaction for miRNA− >TF interactions using high-confidence interactions from the target prediction algorithms: TargetScan, PicTar, miRanda, and RNAHydbrid. They identified 252 high-confidence interactions involving 67 miRNAs and 73 TFs. Graphical analysis of the out-degree and in-degree distribution of this set of interactions revealed that both followed exponential distributions ($R^2 = 0.90$ and $R^2 = 0.84$, respectively). The finding relating to out-degree meant that unlike in transcription networks where some TFs have many targets, it is not possible find any clear miRNA hubs. In transcription networks, knock-out of TF hubs can be lethal to the organism. In contrast, the lack of miRNA hubs supports the suggestion that miRNAs play a role in fine-tuning gene expression [8], rather than acting as master regulators of gene expression. Although the findings suggest an exponential distribution for the number of miRs per TF target (in-degree), when all targets are considered, the findings suggest the in-degree distribution follow a power law. In this distribution, they found gene target hubs are enriched for TFs. This study thus agrees with Shalgi's findings (discussed above).

### 42.4.2 Local Architecture Properties of miRNA-mediated Post-transcriptional Networks

***42.4.2.1 TF-TF, miR-miR, and TF-miR Coordinated Regulation of Gene Targets.*** Coordinated regulation of gene expression programs by multiple transcription elements is a well-recognized property of GRN analysis. In the case of miRNAs, similar research has explored the extent to which miRNAs regulate their targets in pairs, or together with a TF element.

Shalgi *et al.* studied whether there are significantly high numbers of coregulatory miR-miR pairs or TF-miR pairs with a shared gene target among the targets in the PicTar and TargetScan datasets. In the former case, the test statistic was obtained by screening for the co-occurence (and avoidance) of complementary binding sequences in 3'UTRs of target genes for each pair. They identified 107 significant co-occurrence pairs in the TargetScan dataset and 199 significant co-occurrence pairs in the PicTar dataset. They built a miR combinatorial network for each dataset that showed scale-free properties. This meant that a few miRs are highly connected to other miRs, whereas others are much less connected.

Similarly, to study TF-miRNA pairs, they screened for the co-occurrence of a TFBS for the TF in the target gene's promoter using position-specific scoring matrices representing TF binding sites in TRANSFAC and binding seed for the miRNA in the 3'UTR region of the gene. They detected 104 miR-TF pairs in the TargetScan dataset and 916 miR-TF pairs in the PicTar dataset that were significant in two statistical tests. Expression data across human tissues and organs for miRs and mRNAs were used to study the expression coherence of the miR-miR and miR-TF co-occurrence pairs, and it was found that the histogram had peaks in the number of pairs with strong positive and negative correlation coefficients.

Zhou *et al.* analyzed a large integrated network compiled from predicted PicTar and TFBS targets to study the properties of TF-miR, miR-miR, and TF-TF regulatory pairs [79]. They found that TF-TF and miR-miR pairs were more highly abundant than TF-miR pairs in the IRNS. The authors also found that feedforward loops (where the miRNA were suppressed by the TF and many of its targets) were prominent motifs among TF-miR pairs.

Cui *et al.* investigated the extent to which genes that possess features indicating high levels of coordinated regulation by transcription factors also show similarly high levels of coordinated miRNA regulation [17]. They studied the target genes of three TFs, OCT4, NANOG, and SOX2 in human embryonic stem cells (a total of 2046 genes). After dividing the genes into three groups according to whether they are regulated by 1, 2, or all 3 of the TFs, they counted the number of genes that are regulated by miRNAs in each group. They found that miRNA targets are enriched in groups of genes targeted by more TFs.

They then repeated the study on a genome-wide scale by using two datasets of human TFBS to study genes containing more than three (*i.e.*, on average 20 for dataset 1) putative TFBS in the promoter region. They found that the average TFBS-count of the miRNA target genes is significantly higher than that on the non-target genes ($p < 1.9 \times 10^{-55}$) in the first dataset (DS1), with a similar result ($p < 1.5 \times 10^{-216}$) in the second dataset (DS2). Further analysis revealed that there was a high correlation between TFBS-count and miRNA target rate (Pearson's correlation coefficient, $r = 0.9432$ in dataset 1; $r = 0.9680$, DS2), and between miRNA-count and TFBS-count of genes targeted by more than 1 miRNA ($r = 0.7364$, DS1; $r = 0.7200$, DS2). A study of top GO terms for the top 200 genes with both high TFBS-count and miRNA target rate revealed that development categories were enriched among these genes.

Wang *et al.* perform two studies to search for TFs and miRNAs that are most likely to participate in combinatorial regulation of a set of genes that are differentially expressed in a given condition. The two conditions investigated are fetal alcohol syndrome [72] and androgen-independent prostate cancer compared with androgen-dependent prostate cancer [73].

The algorithm involves first using their algorithm *MotifFinder* to identify TFBS and binding sites in the 3'UTRs of genes. They then randomly select a set of TFBSs found in the promoter regions of the genes and binding sites in the 3'UTRs for miRNAs. The algorithm then scores how well a regulatory function based on that set of regulatory binding sites and estimated levels of TFs and miRNAs matches the actual differential regulation for each gene. A fitness score is maintained for each TFBS and miRNA binding site as a cumulative score constructed from the individual TFBS and miRNA binding scores in each of the random sets. Top-scoring TFBS and miRNA binding sites are matched to TFs in TRANSFAC and miRNAs in miRBase.

***42.4.2.2 miR-regulated FFLs, FBLs, and Other 3 or Greater Node-Motifs in Regulatory Networks.*** Theoretical and evidence-based approaches have been used to identify local circuit types that are expected to characterize miRNA-containing networks. Prediction of circuit motif types is often guided by observations of gene expression responses to genetic mutation analysis.

Hornstein and Shomron [32] observed that forward genetic screens that identified loss-of-function in miRNA mutants tended to study miRNA relationships where the miRNA and the target had high binding avidity (multiple binding sites in the target). They postulated that in many miRNA-target relationships, the role of the miRNA was less dramatic.

Gene expression analysis of gene sets enriched for miRNA targets in miRNA overexpression studies revealed that miRNA expression is mutually exclusive with target mRNA transcripts in some tissues spatially or temporally [68]; and that in other organisms and conditions the target may be expressed in the same tissue as the miRNA but at significantly lower levels [21, 67].

Based on these findings, Hornstein and Shomron suggested potential wirings for miRNA circuits that would explain the two different observed responses. They proposed that miRNAs may commonly act within a feedforward loop (FFL), where a TF and a miRNA both regulate a common target, and the TF regulates the miRNA gene. The authors proposed that miRNAs may participate in coherent FFLs (type 3 and 4) (Figure 42.3b and c), which would provide a molecular basis for findings that targets and miRNAs are anticorrelated in expression patterns. In both types of coherent FFL, the miRNA enforces the decision of the TF regulator. In the case of type 3 coherent FFLs, the miRNA is suggested to function by mopping up any leaking basal transcription of a target that was downregulated at the transcription level.

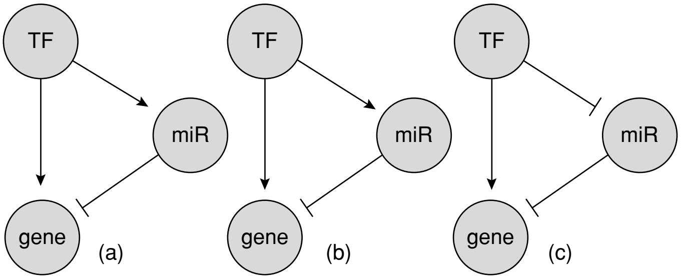


**Figure 42.3** (a) Coherent FFLs. (b),(c) Incoherent FFLs.

The authors suggested a type I incoherent FFL (Figure 42.3a) could underly findings that targets are expressed but at lower levels in miRNA-specific tissue. In this circuit, a transcription regulator activates a target but also upregulates the miRNA, which maintains the target at a fairly constant level. Thus, the miRNA may reduce the effects of noise associated with varying TF levels on the mRNA target.

Shalgi *et al.* studied the frequency of various FFLs in the networks constructed from PicTar and TargetScan datasets [63]. They first searched for "FFL TF $-$ $>$ miR" motifs among TF-miR pairs, by searching for the occurrence of the TFBS in the miR itself among identified TF-miRNA pairs in the earlier study. They found that the frequency of such motifs was significant for the PicTar dataset ($p < 10^{-4}$) but were modest for the TargetScan dataset ($p = 0.024$). Similar studies were performed for FFL miR $-$ $>$TF motifs (where the miR regulates the TF), and results again showed motif overrepresentation for miR-TF pairs in the PicTar (but not the TargetScan) datasets. A composite loop network motif, FFL miR $< --$ $>$ TF (where the miR and the TF regulate eachother), showed overepresentation among pairs from the PicTar dataset. The final motif indirect FFL, which involved one TF exerting regulation of its partner miRNA via another mediator TF, was found to be overrepresented among pairs from both datasets.

Tsang *et al.*, used expression studies to identify whether type I (incoherent) or type II (coherent) FFLs and feedback loops (FBLs) were recurrent across human and mouse tissues [71]. They analyzed the expression levels of 60 human intronic miRNAs (using the expression levels of the host gene as proxy) and thousands of mRNAs in the Novartis human expression atlas and mouse atlas [69]. For each microRNA, they compiled a list of genes ranked according to expression pattern correlation against the miRNA among the arrays. They then searched for predicted targets of the miRNA found using TargetScanS [42] in the list. The authors reasoned that where a predicted target appeared high in the list, it indicated the presence of a type I circuit where the mRNA and miRNA are correlated in expression. Similarly, where the target appeared in a low ranking, it indicated the presence of a type II circuit—the mRNA and its target are anticorrelated.

Among the 60 ranked lists, 75% had a significantly higher number of predicted targets in the top and bottom 10th percentile of the ranked list, suggesting that many miRNAs and targets exist in type I (incoherent) circuits and many in type II (coherent) circuits, respectively. They repeated the study of the lists against a conservation score (CE) for miRNA seed matches in a set of genes, to avoid reliance on noisy target prediction algorithms. For both human and mouse, they found a large percentage of miRNAs possessed genes with a significant CE score in the top ten and bottom ten percentile sets, indicating a type I and type II bias, respectively.

The study was repeated in homogeneous neuronal cell population expression profiles to address the concern that human and mouse datasets contain tissues that may contain mixed cell types.

The study was conducted for miRNAs in three types of motor neurons in a developmental time course set and in the profiles of 12 homogenous neuronal cell types (mature cells). Again, both sets displayed a bias for type I and II circuits. Mature neuronal cells displayed a stronger bias for the type I circuit over the type II circuit

(44% vs 17%). This lends support to suggestions that miRNAs may have a different function in neuronal cells than in non-neuronal cells, but only with respect to mature neuronal cells.

These findings suggested to the authors that the miRNA function is controlled by an activation preference for distinct circuit types in different cell types. Findings of prevalence of type II circuits confirm the findings of Sood *et al.* [67] and others that targets are expressed at lower levels in tissues where the miRNA gene is expressed. On the other hand, type I circuits are less supported with experimental data. The authors suggest type I circuits play a role in permitting a fast response to network component fluctuations to ensure homeostasis and avoid random switching events.

Martinez *et al.* analyzed a real genome-scale miRNA GRN in *C. elegans* that integrates TF $->$ miR interactions detected with Y1H assays and high-confidence miR $->$ TF predictions, to study the presence of FBLs. They first study the presence of type I miRNA $<->$ TF composite FBLs, identifying 23 type I FBLs involving 14 miRNAs and 16 TFs. These comprised both single-negative and double-negative FBLs. The TF $->$ miR interactions in several motifs were confirmed *in vivo* using TaqMan PCR assays, whereas the miR $->$ TF interactions were confirmed with Western blotting. The authors also identified higher order composite FBLs, for example, type II FBLs containing 1 miRNA and 2 TFs.

They found composite miRNA $->$ TF FBLs occurred twice as frequently in the integrated miRNA GRN than in random background networks generated with three different randomization methods. They found that miRNAs and TFs in the integrated miRNA GRN that participated in FBLs had a higher in-degree and out-degree. They further found that nodes with a high flux capacity ($F_c = k_{\text{in}} \times k_{\text{out}}$) were likely to participate in FBLs. However, noting that FBLs remain overrepresented even against a background network generated with the randomization method involving edge switching where individual node degree and overall degree distribution remains unaltered (and thus $F_c$ remains the same), they conclude high flux capacity is not the only determinant in the presence of FBLs in the real network.

Yu *et al.* examined system-wide motifs involving miRNAs and TFs in integrated networks constructed from PicTar and miRanda datasets. The authors exhaustively enumerated a list of possible 3-node subgraphs to be studied in the IRNs, which included instances where the miRNA or TF were regulatory targets (not previously explored), yielding 46 motifs in total (compared with the 5 motif types studied by Shalgi *et al.*, discussed above). For each miRNA, the authors counted the occurrence of the miRNA in each subgraph type in the two IRNs, and evaluated this count as a z-score against the expected number of occurrences of the miRNAs in the subgraph type, derived from the overall representation of the subgraph type in the IRN.

The authors then clustered the motifs and top-scoring subgraphs were selected based on z-score and cluster number. High-scoring subgraphs included regulated FBL where two TFs regulate each other and one miRNA regulates both of them (z-score: 68.1 for miRanda, 24.4 for Pictar), and coregulation motifs where miRNAs and TFs coregulate a common target without coregulating each other (z-score: 38.1 for miRanda, 50.2 for Pictar). miRNA-associated FFLs were also high-scoring subgraphs.

The authors also examined the functionality of the regulated FBL using mathematical modeling to determine the effect of miR regulatory strength and different initial molecular levels on system properties such as state transition.

The authors clustered miRNAs according to similarity of network motif profiles, and found that there were two distinct classes of miRNAs: class I miRNAs that were enriched for network subgraphs in which the miRNAs were regulated by many TFs, and class II miRNAs that were enriched for network subgraphs in which the miRNAs were only regulated by one or a few TFs. The authors used experimental datasets on whole genome *in vivo* binding of human TFs to support their findings. They also studied the gene expression of miRNAs in tissues in both embryonic development stages and adult tissues across several organisms. They found that class I miRNAs were enriched in embryonic developmental conditions, whereas class II miRNAs were enriched in the adult tissues, suggesting that different TFs may be responsible for regulating the miRs at different stages during development.

Zhou *et al.* investigated local properties of combinatorial networks in *A. thaliana* [79]. They first identified TF-miRNA coordinating pairs by the presence of a binding site in the promoter region of the miRNA target, with additional requirements such as coherent expression patterns of targets. Local motifs were further detected including feedforward, double feedforward, composite pair, and composite loop networks. The statistical significance of the co-occurence of the miRNA and a motif in the promoter was determined using the cumulative hypergeometric distribution. The statistical significance of the other network motifs was obtained for the corresponding number of coordinating pairs using a Monte Carlo simulation.

Table 42.1 summarizes popular miRNA-containing motifs in networks constructed using various methods.

***42.4.2.3 Role of miRNAs in Targeting Signaling Network Motifs.*** Cui *et al.* investigated the nature of proteins coded for by genes that are predicted targets of miRNAs (using PicTar and TargetScanS), which are found in signaling networks [16]. In particular, they explored proteins that play a role in a signaling network in a mammalian hippocampal CA1 neuron (Ma'ayan *et al.*) that contained commonly used pathways in many cell types. After grouping proteins into categories from upstream (*e.g.*, ligands) to downstream (*e.g.*, nuclear proteins) groups, they observed that the ratio of miRNA targets to other proteins in the group increases with signal information flow from the upstream to the downstream. They then studied the nature of adaptor proteins that are thought to play a role in regulating downstream signaling proteins. After dividing adaptor proteins into high-link (bind to >4 downstream components) and low-link adaptors, they observed that high-link adaptors had a higher fraction of genes that were miRNA targets than low-link adaptors. The authors suggest miRNAs may control levels of the downstream components, to confer precision to the cellular response to stimuli mediated by high-link adaptors.

They also analyzed the abundance of miRNA targets among 11 different classes of network motifs identified in the network. They classified the motifs based on the number of nodes that are miRNA targets (0–3 for 3-node motifs or 0–4 for 4-node motifs). They then calculated a score for each motif as a ratio of positive links to

**Table 42.1 Examples of some significant miRNA-containing motifs discovered in various constructed miRNA-mediated networks**

| Motif | Author | Description of Network where Motif was Found | Method | Potential Role |
|---|---|---|---|---|
| regulated FBL | Yu *et al.* | integrated regulatory network formed with PicTar and miRanda | motif count and statistical analysis | canalization of development. |
| miR-TF regulatory pairs | Yu *et al.* | integrated regulatory network formed with PicTar and miRanda | motif count and statistical analysis | canalization of development. |
| type I FFL/FBL | Tsang *et al.* | Gene expression network across multiple conditions and in mature neuronal cells in mouse | find target enrichment in top %10 of list of genes ranked according to correlation with miRNA | noise reduction and target stabilization. |
| type II FFL/FBL | Tsang *et al.* | Gene expression network across multiple conditions and in developing mouse neuronal cells | find target enrichment in bottom %10 of list of genes ranked according to correlation with miRNA | reinforce decision of principal regulator. |
| FFL/FBL (TF –> miR) | Shalgi *et al.* | IRN built with PicTar and TargetScanS (motif score less significant) | motif count among TF-miR pairs and statistical analysis | — |
| FFL/FBL (miR –> TF) | Shalgi *et al.* | IRN built with PicTar and TargetScanS (motif score less significant) | motif count among TF-miR pairs and statistical analysis | — |
| double-negative FBL | Martinez *et al.* | Real integrated miRNA network in *C. elegans* | motif count and statistical analysis | lead to bistability and act as toggle-switch. |

total direction links (Ra score) in each subgroup. They found that for most motifs, the Ra score in subgroups with 0 nodes was less than the average Ra across all motifs; and, for most motifs, the ratio of positive links in the subgroups increased with the number of miRNA targets. Thus, they discovered for most motifs (in particular, highly abundant motifs), a clear positive correlation between positive link ratio and miRNA target number in the motif.

They then studied the miRNA-targeted scaffold proteins, which play a role in recruiting distinct proteins to different pathways. They found that those proteins that were miRNA targets were more likely to form a motif than were other scaffold proteins. They also found that highly linked scaffold proteins were targeted by multiple miRNAs (*e.g.*, 5 or 6 miRNAs for SNAP25 and CRK, respectively). This is consistent with the highly specialized spatio-temporal expression behavior of scaffold proteins.

They then studied the fraction of miRNA targets among shortest-path proteins in five distinct cellular machines, which may be activated by the signaling pathway. They found that these proteins possess a lower fraction of miRNA targets than in the rest of the network, suggesting miRNAs avoid disturbing basic cellular processes mediated by proteins that are shared among basic cellular machines. Overall, they argue that these patterns are suggestive of a role for miRNAs in terminating incoming messages and in permitting quick and robust transitions.

## 42.5 CONCLUSION

Identification of microRNA-mediated networks and analysis of network properties is a growing field. As we identify miRNA-target interactions, we are able to integrate circuits constructed from these interactions with validated TF-mediated circuits to achieve a more detailed picture of regulation of cell behavior at multiple levels.

The largest body of research into properties of miRNA behavior has focused on identifying novel miRNA genes and their direct targets. Predictions of miRNA-target relationships continue to be useful in understanding miRNA behavior; however, such tools must be used with caution. Target prediction data are therefore increasingly used in conjunction with gene expression and proteomic data to learn functional targets and infer biological consequences of different levels of repression.

Studies that aim to characterize the miRNA transcript reveal important information regarding the nature of miR regulation and likely regulators of miRNA genes. Further studies into properties of miRNA transcripts will be assisted by improved EST collection and annotation. Further information regarding the transcriptome will be aided by the availability of modern transcriptome sequencing tools such as Illumina sequencing technology.

Forward-engineering remains a reliable way to infer the existence miRNA-containing circuits that perform signal processing functions within the cell. Reverse engineering of modules reveals miRNAs and genes showing similar properties. The study of the network motifs, including increasingly wide-ranging and sophisticated patterns, in miRNA-containing networks continues to provide insight into possible roles for miRs in different conditions and tissues.

## REFERENCES

1. I. Alvarez-Garcia and E.H. Miska. MicroRNA functions in animal development and human disease. *Development*, 132(21):4653–4662, 2005.
2. V. Ambros. The function of animal MicroRNAs. *Nature*, 431:350–355, 2004.

3. M.M. Babu, N.M. Luscombe, L. Aravind, M. Gerstein, and S.A. Teichmann. Structure and evolution of transcriptional regulatory networks. *Curr Opin Struct Biol*, 14:283–291, 2004.

4. S. Bagga, J. Bracht, S. Hunter, K. Massirer, J. Holtz, *et al.* Regulation by let-7 and lin-4 miRNAs results in target mRNA degradation. *Cell*, 122:553–563, 2005.

5. T.L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Mach Learn*, 21:51–83, 1995.

6. A.L. Barábasi and Z.N. Oltvai. Network Biology: Understanding the cell's functional organization. *Nat Genet*, 5:101–113, 2004.

7. D.P. Bartel. MicroRNAs: Genomics, biogenesis, mechanism, and function. *Cell*, 116:281–297, 2004.

8. D.P. Bartel and C.Z. Chen. Micromanagers of gene expression: The potentially widespread influcence of metwazoan microRNAs. *Nat Rev Genet*, 5:396–400, 2004.

9. Z. Bar-Joseph, G.K. Gerber, T.I. Lee, N.J. Rinaldi, J.Y. Yoo, F. Robert, D.B. Gordon, E. Fraenkel, T.S. Jaakkola, R.A. Young, and D.K. Gifford. Computational discovery of gene modules and regulatory networks. *Nat Biotechnol*, 21:1337–1342, 2003.

10. I. Bentwich, A. Avniel, Y. Karov, R. Aharonov, S. Gilad, O. Barad, A. Barzilai, P. Einat, U. Einav, E. Meiri, *et al.* Identiifcation of hundreds of conserved and nonconserved human microRNAs. *Nat Genet*, 37:766–770, 2005.

11. E. Berezikov, V. Guryev, J. Van De Belt, E. Wienholds, R.H. Plasterk, and E. Cuppen. Phylogenetic shadowing and computational identification of human microRNA genes. *Cell*, 120:21–24, 2005.

12. R. Bonneau, D. Reiss, P. Shannon, M. Facciotti, L. Hood, N. Baliga, and V. Thorsson. The inferelator: An algorithm for learning parsimonious regulatory networks from systems—biology data sets de novo. *Genome Biol*, 7:R36, 2006.

13. J.C. Carrington and V. Ambros. Role of microRNAs in plant and animal development. *Science*, 301:336–338, 2003.

14. K. Chaudhuri and R. Chatterjee. MicroRNA detection and target prediction: Integration of computational and experimental approaches. *DNA Cell Biol*, 26:5, 2007.

15. C. Cheng and L.M. Li. Inferring microRNA activities by combining gene expression with microRNA target prediction. *PLoS Comput Biol*, 3(4):e1989, 2008.

16. Q. Cui, Z. Yu, E. Purisima, and E. Wang. Principles of microRNA regulation of a human cellular signaling network. *Mol Syst Biol*, 2:46, 2006.

17. Q. Cui, Z. Yu, Y. Pan, E. Purisima, and E. Wang. MicroRNAs preferentially target the genes with high transcriptional regulation complexity. *Biochem Biophys Res Commun*, 352:733–738, 2007.

18. P. D'Haeseleer, X. Wen, S. Fuhrman, and R. Somogyi. Linear modeling of mRNA expression levels during CNS development and injury. *Pacific Symposium Biocomputing*, 1999, pp. 41–52.

19. H. De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *J Comput Biol*, 9:67–103, 2002.

20. D.V. Dugas and B. Bartel. MicroRNA regulation of gene expression in plants. *Curr Opin Plant Biol*, 7:512–520, 2004.

21. K.K. Farh, A. Grimson, C. Jan, B.P. Lewis, and W.K. Johnston. The widespread impact of mammalian microRNAs on mRNA repression and evolution. *Science*, 310:1817–1821, 2005.

22. A.S. Flynt and E.C. Lai. Biological principles of microRNA-mediated regulation: Shared themes amid diversity. *Nat Rev Genet*, 9:831–842, 2008.

23. N. Friedman, N. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *J Comput Biol*, 7:601–620, 2000.

24. S. Fujita and H. Iba. Putative promoter regions of miRNA genes involved in evolutionary conserverd regulatory systems. *Bioinformatics*, 24(3):303–308, 2008.

25. S. Griffiths-Jones. The microRNA Registry. *Nucleic Acids Res*, D109–D111, 2004.

26. S. Griffith-Jones, R.J. Grockock, S. van Dongen, A. Bateman, and A.J. Enright. miRBase:microRNA sequenes, targets and gene nomenclature. *Nucleic Acids Res*, 34:D140–D144, 2006.

27. S. Griffiths-Jones, H.K. Saini, S. van Dongen, and A.J. Enright. *Nucleic Acids Res*, 36(Database Issue):D154–D158, 2008.

28. D. Grün and N. Rajewsky. Chapter 12: Computational prediciton of microRNA targets in vertebrates, fruitfiles and nematodes. In K. Appasani, editor, *MicroRNAs: From Basic Science to Disease Biology*, Cambridge University Press, Cambridge, U.K., 2007.

29. J. Gu, T. He, Y. Pei, F. Li, X. Wang, J. Zhang, X. Zhang, and Y. Li. Primary transcripts and expressions of mammal intergenic microRNAs detected by mapping ESTs to their flanking sequences. *Mamm Genome*, 17:1033–1041, 2006.

30. N. Guelzim, S. Bottani, P. Bourgine, and F. Kepes. Topological and causal structure of the yeast transcriptional regulatory network. *Nat Genet*, 31:60–63, 2002.

31. O. Hobert. Common logic of transcription factor and microRNA action. *Trends Biochem Sci*, 29(9):462–468, 2004.

32. E. Hornstein and N. Shomron. Canalization of development by microRNAs. *Nat Genet Suppl*, 38:s20, 2006.

33. J.C. Huang, Q.D. Morris, and B.J. Frey. Detecting microRNA targets by linking sequence, microRNA and gene expression data. *Lect Notes Comput Sci*, 3909:114–129, 2006.

34. J.C. Huang, Q.D. Morris, and B.J. Frey. Bayesian inference of microRNA targets from sequence and expression data. *J Comput Biol*, 14(5):550–563, 2007.

35. R.J. Johnston, Jr., S. Chang, J.F. Etchberger, C.O. Ortiz, and O. Hobert. MicroRNAs acting in a double-negative feedback loop to control a neuronal cell fate decision. *Proc Natl Acad Sci U S A*, 102:12449–12454, 2005.

36. J.G. Joung, K.B. Hwang, J.W. Nam, S.J. Kim, and B.T. Zhang. Discovery of microRNA-mRNA modules via population-based probabilistic learning. *Bioinformatics*, 23(9):1141–1147, 2007.

37. J.G. Joung and Z. Fei. Identification of microRNA regulatory modules in Arabidopsis via a probabilistic graphical model. *Bioinformatics*, 25:387–393, 2009.

38. G. Karlebach and R. Shamir. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology*, 9:770–780, 2008.

39. A. Krek, D. Grün, M.N. Poy, *et al.* Combinatorial microRNA target predictions. *Nat Genet*, 37:495–500, 2005.

40. J. Lee, Z. Li, R. Brower-Sinning, and J. Binno. Regulatory circuit of human microRNA biogenesis. *PLoS Comput Biol*, 3(4):e67, 2007.

41. B.P. Lewis, I.H. Shih, M.W. Jones-rhoades, D.P. Bartel, and C.B. Burge. Prediction of mammalian microRNA targets. *Cell*, 115:787–798, 2003.

42. B.P. Lewis, C.B. Burge, and D.P. Bartel. Conserved seed pairing, often flanked by adenosines, indicates that thousands of human genes are microRNA targets. *Cell*, 120:15–20, 2005.

43. W.S. Leung, S.M. Yiu, D.W. Cheung, L. Lai, M.C. Lin, and H.F. King. Computational prediction on mammalian and viral miRNAs—A review. *Int J Integr Biol*, 1(2):118–126, 2007.

44. X. Li and R.W. Carthew. A microRNA mediates EGF receptor signaling and promotes photoreceptor differentiation in the *Drosophila* eye. *Cell*, 123:1267–1277, 2005.

45. L.P. Lim, N.C. Lau, and P. Garrett-Engele. Microarray analysis shows that some microRNAs downregulate large numbers of target mRNAs. *Nature*, 433:769–733, 2005.

46. B. Liu, J. Li, and A. Tsykin. Discovery of functional miRNA-mRNA regulatory modules with computational methods. *J Biomed Inform*, 42(4):685–691, 2009.

47. J. Lu, G. Getz, E.A. Miska, E. Alvarez-Saavedra, J. Lamb, D. Peck. A. Sweet-Cordero, B.L. Ebert, R.H. Mak, A.A. Ferrando, *et al.* MicroRNA expression profiles classify human cancers. *Nature*, 435:834–838, 2005.

48. V. Matys, O.V. Kel-Margoulis, E. Fricke, I. Liebich, S. Land *et al.* TRANSFAC and its module TRANSCompel: Transcriptional gene regulation in eukaryotes. *Nucleic Acids Res*, 34:D108–D110, 2006.

49. J.S.M. Mattick. A new paradigm for developmental biology. *J Exp Biol*, 210:1526–1547, 2007.

50. M. Megraw, V. Baev, V. Rusinov, *et al.* MicroRNA promoter element discovery in Arabidopsis. *RNA*, 12:1612–1619, 2006.

51. K. Murphy and S. Mian. *Modeling gene expression data using dynamic bayesian networks*, Technical Report, Berkely, CA Computer Science Division, University of California, 1999.

52. U. Ohler, S. Yekta, L.P. Lim, D.P. Bartel, and C.B. Burge. Patterns of flanking sequence conservation and a characteristic upstream motif for microRNA gene identification. *RNA*, 10:1309–1322, 2004.

53. K.A. O'Donnell, E.A. Wentzel, K.I. Zeller, C.V. Dang, and J.T. Mendell. c-Myc-regulated microRNAs modulate E2F1 expression. *Nature*, 435:839–843, 2005.

54. N. Rajewsky. MicroRNA target predictions in anmials. *Nat Genet Suppl*, 38:S8–S13, 2006.

55. W. Ritchie, S. Flamant, and E.J. Rasko. Predicting microRNA targets and functions: Traps for the unwary. *Nat Methods*, 6(6):397–398, 2009.

56. H.K. Saini, S. Griffiths-Jones, and A.J. Enright. Genomic analysis of human microRNA transcripts. *PLoS Comput Biol*, 104(45):17719–17724, 2007.

57. H. Saini, A.J. Enright, and S. Griffiths-Jones. Annotation of mammalian primary microRNAs. *BMC Genom*, 9:564, 2008.

58. E. Segal, M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman. Module networks: Identifying regulatory modules and their condition-specific regulators from gene expression data. *Nat Genet*, 34:166–176, 2003.

59. P. Sethupathy, M. Megraw, M.I. Barrasa, and A.G. Hatzifeorgiou. Computational identification of regulatory factors involved in microRNA transcription. *Lect Notes Comput Sci*, 3746:457–468, 2005.

60. P. Sethupathy, M. Megraw, and A.G. Hatzigeorgiou. A guide through present computational approaches for the identification of mammalian microRNA targets. *Nat Methods*, 3(11):881–886, 2006.

61. P. Sethupathy, B. Corda, and A.G. Hatzigeorgiou. TarBase: A comprehensive database of experimentally supported animal microRNA targets. *RNA*, 12:192–197, 2006.

62. P. Sethupathy, M. Megraw, and A.G. Hatzigeorgiou. Computational approaches to elucidate miRNA biology. In K. Appasani, editor, *MicroRNAs: From Basic Science to Disease Biology*. Cambridge University Press, Cambridge, U.K., 2007.

63. R. Shalgi, D. Lieber, M. Oren, and R. Pilpel. Global and local architecture of the mammalian microRNA-transcription factor regulatory network. *PLoS Comput Biol*, 3(7):e131, 2007.

64. S.S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of *Escherichia coli*. *Nat Genet*, 31:64–68, 2002.

65. I. Shmulevich, E.R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.

66. S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. *ProC Int Conf Intell Syst Mol Biol*, volume 8, 2000, pp. 344–354.

67. P. Sood, A. Krek, M. Zavolan, G. Macino, and N. Rajewsky. Cell-type-specific signatures of microRNAs on target mRNA expression. *Proc Natl Acad Sci U S A*, 103:2746–2751, 2006.

68. A. Stark, J. Brennecke, N. Bushati, R.B. Russell, and S.M. Cohen. Animal microRNAs confer robustness to gene expression and have a significant impact on 3UTR evolution. *Cell*, 123:1133–1146, 2005.

69. A.I. Su, T. Wiltshire, S. Batalov, H. Lapp, K.A. Ching, D. Block, J. Zhang, R. Soden, M. Hayakawa, G. Kreiman, *et al.* A gene atlas of the mouse and human protein-encoding transcriptomes. *Proc Natl Acad Sci U S A*, 101:6062–6067, 2004.

70. D.H. Tran, K. Satou, and T.B. Ho. Finding microRNA regulatory modules in human genome using rule induction. *BMC Bioinformatics*, 9(Suppl 12):S5, 2008.

71. J. Tsang, J. Zhu, and A. van Oudenaarden. MicroRNA-mediated feedback and feedforward loops are recurrent network motifs in mammals. *Mol Cell*, 26:753–767, 2007.

72. G. Wang, X. Wang, Y. Wang, J.Y. Yang, L. Li, K.P. Nephew, H.J. Edenberg, F.C. Zhou, and Y. Liu. Identification of transcription factor and microRNA binding sites in responsible to fetal alcohol syndrome. *BMC Genomics*, 9(Suppl 1):S19, 2008.

73. G. Wang, Y. Wang, W. Feng, X. Wang, J.Y. Yang, Y. Zhao, Y. Wang, and Y. Liu. Transcription factor and microRNA regulation in androgen-dependent and -independent prostate cancer cells. *BMC Genomics*, 9(Suppl 2):S22, 2008.

74. X. Xie, E.J. Kulbokas, T.R. Golub, *et al.* Systematic discovery of regulatory motifs in human promoters and 3'UTRs by comparison of several mammals. *Nature*, 434:338–345, 2005.

75. Z. Xie, E. Allen, N. Fahlgren, A. Calamar, S.A. Givan, and J.C. Carrington. Expression of Arabidopsis miRNA genes. *Plant Physiol*, 138:2145–2154, 2005.

76. S. Yoon and G. De Micheli. Prediction of regulatory moduels comprising microRNAs and target genes. *Bioinformatics*, 21:ii93–ii100, 2005.

77. C.B.-Z. Zilberstein, M. Ziv-Ukelson, R.Y. Pinter, and Z. Yakhini. A high-throughput approach for associating microRNAs with their activity conditions. *J Comput Biol*, 13:245–266, 2006.

78. X. Zhou, J. Ruan, G. Wang, and W. Zhang. Characterization and identification of microRNA core promoters in four model species. *PLoS Comput Biol*, 3:e37, 2007.

79. X. Zhou and W. Zhang. Combinatory circuits of miRNAs and transcription factors in plant gene regulations. *RECOMB Satellite Conference on System Biology*, 2007.

80. Y. Zhou, J. Ferguson, J.T. Chang, and Y. Kluger. Inter- and intra-combinatorial regulation by transription factors and microRNAs. *BMC Genomics*, 8:396, 2007.

# INDEX

*Algorithms in Computational Molecular Biology,* Edited by Mourad Elloumi and Albert Y. Zomaya

Wiley Series on

**Bioinformatics: Computational Techniques and Engineering**

Bioinformatics and computational biology involve the comprehensive application of mathematics, statistics, and computer science to the understanding of living systems. Research and development in these areas require cooperation among specialists from the fields of biology, computer science, mathematics, statistics, physics, and related sciences. The objective of this book series is to provide timely treatments of the different aspects of bioinformatics spanning theory, new and established techniques, technologies and tools, and application domains. This series emphasizes algorithmic, mathematical, statistical, and computational methods that are central in bioinformatics and computational biology.

Series Editors: **Professor Yi Pan** and **Professor Albert Y. Zomaya**
pan@cs.gsu.edu albert.zomaya@sydney.edu.au

---

*Knowledge Discovery in Bioinformatics: Techniques, Methods, and Applications*
Xiaohua Hu and Yi Pan

*Grid Computing for Bioinformatics and Computational Biology*
Edited by El-Ghazali Talbi and Albert Y. Zomaya

*Bioinformatics Algorithms: Techniques and Applications*
Ion Mandiou and Alexander Zelikovsky

*Analysis of Biological Networks*
Edited by Björn H. Junker and Falk Schreiber

*Computational Intelligence and Pattern Analysis in Biological Informatics*
Edited by Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Jason T. L. Wang

*Mathematics of Bioinformatics: Theory, Practice, and Applications*
Matthew He and Sergey Petoukhov

*Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications*
Edited by Mourad Elloumi and Albert Y. Zomaya