BIOINFORMATICS FOR BIOLOGISTS

The computational education of biologists is changing to prepare students for facing the complex data sets of today's life science research. In this concise textbook, the authors' fresh pedagogical approaches lead biology students from first principles towards computational thinking.

A team of renowned bioinformaticians take innovative routes to introduce computational ideas in the context of real biological problems. Intuitive explanations promote deep understanding, using little mathematical formalism. Self-contained chapters show how computational procedures are developed and applied to central topics in bioinformatics and genomics, such as the genetic basis of disease, genome evolution, or the tree of life concept. Using bioinformatic resources requires a basic understanding of what bioinformatics is and what it can do. Rather than just presenting tools, the authors – each a leading scientist – engage the students' problem-solving skills, preparing them to meet the computational challenges of their life science careers.

PAVEL PEVZNER is Ronald R. Taylor Professor of Computer Science and Director of the Bioinformatics and Systems Biology Program at the University of California, San Diego. He was named a Howard Hughes Medical Institute Professor in 2006.

RON SHAMIR is Raymond and Beverly Sackler Professor of Bioinformatics and head of the Edmond J. Safra Bioinformatics Program at Tel Aviv University. He founded the joint Life Sciences – Computer Science undergraduate degree program in Bioinformatics at Tel Aviv University.

BIOINFORMATICS FOR BIOLOGISTS

EDITED BY
Pavel Pevzner
University of California, San Diego, USA

AND
Ron Shamir
Tel Aviv University, Israel

CAMBRIDGE UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town,
Singapore, São Paulo, Delhi, Tokyo, Mexico City

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9781107011465

First published 2011

Printed in the United Kingdom at the University Press, Cambridge

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

Library of Congress Cataloging in Publication data
Bioinformatics for biologists / edited by Pavel Pevzner, Ron Shamir.
p. cm.
Includes index.
ISBN 978-1-107-01146-5 (hardback)
1. Bioinformatics. I. Pevzner, Pavel. II. Shamir, Ron.
QH324.2.B5474 2011
572.8 – dc23 2011022989

ISBN 978-1-107-01146-5 Hardback
ISBN 978-1-107-64887-6 Paperback

To Ellina, the love of my life.
(P.P.)

To my parents, Varda and Raphael Shamir.
(R.S.)

CONTENTS

EXTENDED CONTENTS

PREFACE

What is this book?

This book aims to convey the fundamentals of bioinformatics to life science students and researchers. It aims to communicate the computational ideas behind key methods in bioinformatics to readers without formal college-level computational education. It is not a "recipe book": it focuses on the *computational ideas* and avoids technical explanation on running bioinformatics programs or searching databases. Our experience and strong belief are that once the computational ideas are grasped, students will be able to use existing bioinformatics tools more effectively, and can utilize their understanding to advance their research goals by envisioning new computational goals and communicating better with computational scientists.

The book consists of self-contained chapters each introducing a basic computational method in bioinformatics along with the biological problems the method aims to solve. Review questions follow each chapter. An accompanying website (www.cambridge.org/b4b) containing teaching materials, presentations, questions, and updates will be of help to students as well as educators.

Who is the audience for the book?

The book is aimed at life science undergraduates; it does not assume that the reader has a background in mathematics and computer science, but rather introduces mathematical concepts as they are needed. The book is also appropriate for graduate students and researchers in life science and for medical students. Each chapter can be studied individually and used individually in class or for independent reading.

Why this book?

In 1998, Stanford professor Michael Levitt reflected that computing has changed biology forever, even if most biologists did not know it yet. More than a decade later, many biologists have realized that computational biology is as essential for this century's biology as molecular biology was in the last century. Bioinformatics[1] has become an essential part of modern biology: biological research would slow down dramatically if one suddenly withdrew the modern bioinformatics tools such as BLAST from the arsenal of biologists. We cannot imagine forward-looking biological research that does not use any of the vast resources that bioinformatics researchers have made available to the biomedical community.

Bioinformatics resources come in two flavors: databases and algorithms. Thousands of databases contain information about protein sequences and structures, gene annotations, evolution, drugs, expression profiles, whole genomes and many more kinds of biological data. Numerous algorithms have been developed to analyze biological data, and software implementations of many of these algorithms are available to biologists. Using these resources effectively requires a basic understanding of what bioinformatics is and what it can do: what tools are available, how best to use them and to interpret their results, and more importantly, what one can reasonably hope to achieve using bioinformatics even if the relevant tools are not yet available.

Despite this richness of bioinformatics resources and methods, and although sophisticated biomedical researchers draw on these resources extensively, the exposure of undergraduates in biology and biochemistry, as well as of medical students, to bioinformatics is still in its infancy. The computational education of biologists has hardly changed in the last 50 years. Most universities still do not offer bioinformatics courses to life sciences undergraduates, and those that do offer such courses struggle with the question of how and what to teach to students with limited computational culture. In the absence of any preparation in computer science, the generation of biologists that went to universities in the last decade remains poorly prepared for the computational aspects of work in their own discipline in the decades to come. Similarly, medical doctors (who will soon have to analyze personal genomes or blood tests that report thousands of protein levels) are not prepared to meet the computational challenges of future medicine.

Biomedical students typically have a very basic computational background, which leads to a serious risk that bioinformatics courses – when offered – will become technical and uninspired. The software tools are often taught and then used as "black

[1] Here and throughout the book, we use the terms bioinformatics and computational biology interchangeably.

boxes," without deeper understanding of the algorithmic ideas behind them. This can lead to under-utilization or over-interpretation of the results that such black-box use produces. Moreover, the students who study bioinformatics at this level will have a much smaller chance of coming up with computational ideas later in their careers when they carry out their own biomedical research. It is therefore essential, in our opinion, that biologists be exposed to deep algorithmic ideas, both in order to make better use of available tools that rely on these ideas, and in order to be able to develop novel computational ideas of their own and communicate effectively with computational biologists later in their careers.

We and others have argued for a revolution in computational education of biologists[2] and noted that the mathematical and computational education of other disciplines have already undergone such revolutions with great success. Physicists went through a computational revolution 150 years ago, and economists have dramatically upgraded their computational curriculum in the last 20 years. As a result, paradoxically, the students in these disciplines are much better prepared for the computational challenges of modern biomedical research than are biology students. Moreover, whatever little mathematical background biologists have, it is mainly limited to classical *continuous* mathematics (such as Calculus) rather than *discrete* mathematics and computer science (e.g. algorithms, machine learning, etc.) that dominate modern bioinformatics. In 2009 we thus came up with a radical prophecy[3] that the education of biologists will soon become as computationally sophisticated as the education of physicists and economists today. As implausible as this scenario looked a few years ago, leading schools in bioinformatics education (such as Harvey Mudd or Berkeley) are well on the way towards this goal.

The time has come for biology education to catch up. Such change may require revising the contents of basic mathematical courses for life science college students, and perhaps updating the topics that are taught. Students' understanding of bioinformatics will benefit greatly from such a change. In parallel, dedicated bioinformatics classes and courses should be established, and textbooks appropriate for them should be developed.

Most undergraduate bioinformatics programs at leading universities involve a grueling mixture of biological and computational courses that prepare students for subsequent bioinformatics courses and research. As a result, some undergraduate bioinformatics courses are too complex even for biology graduate students, let alone

[2] W. Byalek and D. Botstein. Introductory science and mathematics education for 21st-Century biologists. *Science*, 303:788–790, 2004.
P. A. Pevzner. Educating biologists in the 21st century: Bioinformatics scientists versus bioinformatics technicians. *Bioinformatics*, 20:2159–2161, 2004.

[3] P. A. Pevzner and R. Shamir. Computing has changed biology – Biology education must catch up. *Science*, 325:541–542, 2009.

undergraduates. This causes a somewhat paradoxical situation on many campuses today: bioinformatics courses are available, but they are aimed at *bioinformatics* undergraduates and are not suitable for *biology* students (undergraduate or graduate). This leads to the following challenge that, to the best of our knowledge, has not yet been resolved:

Pedagogical Challenge. *Design a bioinformatics course that (i) assumes minimal computational prerequisites, (ii) assumes no knowledge of programming, and (iii) instills in the students a meaningful understanding of computational ideas and ensures that they are able to apply them.*

This challenge has yet to be answered, but we claim that many ideas in bioinformatics can be explained at an intuitive level that is often difficult to achieve in other computational fields. For example, it is difficult to explain the mathematics behind the Ising model of ferromagnetism to a student with limited computational culture, but it is quite possible to introduce the same student to the algorithmic ideas (Euler theorem and de Bruijn graphs) behind the genome assembly. Thus, we argue that the recreational mathematics approach (so brilliantly developed by Martin Gardner and others) coupled with biological insights is a viable paradigm for introducing biologists to bioinformatics. This book is an initial step in that direction.

What is in the book?

Each chapter describes the biological motivation for a problem and then outlines a computational approach to addressing the problem. Chapters can be read separately, as each introduces any needed computational background beyond basic college-level knowledge.

The range of biological topics addressed is quite broad: it includes evolution, genomes, regulatory networks, phylogeny, and more. The computational techniques used are also diverse, from probability and graphs, combinatorics and statistics to algorithms and complexity. However, we made an effort to keep the material accessible and avoid complex computational details (those can be filled in by the interested reader using the references). Figure 1 aims to show for each chapter the biological topics it touches upon and the computational areas involved in the analysis. Naturally, many chapters involve multiple biological and computational areas. Not surprisingly, evolution plays a role in almost all the topics covered, following the famous quote from Theodosius Dobzhansky, “Nothing in biology makes sense except in the light of evolution.”

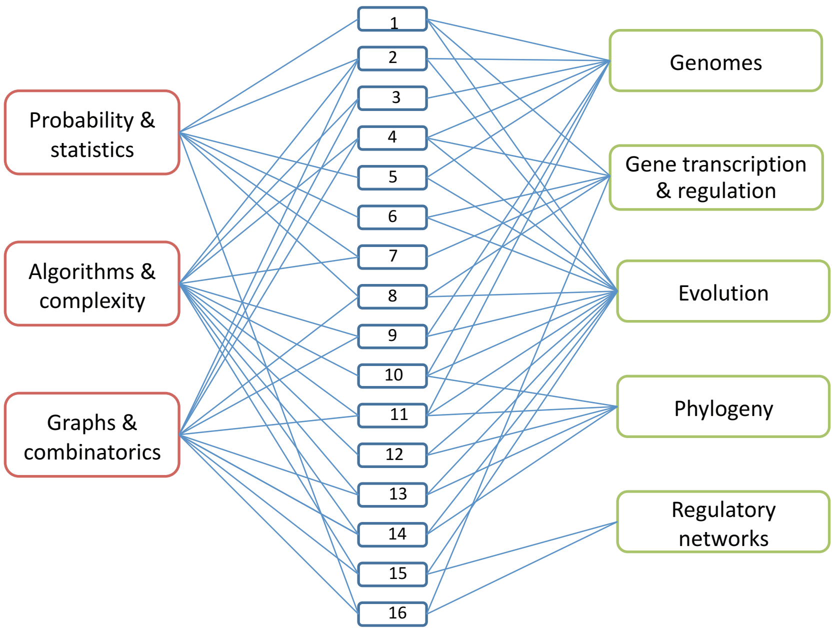

Figure 1 The connections between biological and computational topics for each chapter. The nodes in the middle are chapters, and edges connect each chapter to the biological topics it covers (right) and to the computational topics it introduces (left).

The pedagogical approach, the style, the length, and the depth of the introduced mathematical concepts vary greatly from chapter to chapter. Moreover, even the notation and computational framework describing the same mathematical concepts (e.g. graph theory) across different chapters may vary. As computer scientists say, this is not a bug but a feature: we provided the contributors with complete freedom in selecting the approach that fits their pedagogical goal the best. Indeed, there is no consensus yet on *how* to introduce computer science to biologists, and we feel it is important to see how leading bioinformaticians address the same pedagogical challenge.

How will this book develop?

"Bioinformatics for Biologists" is an evolving book project: we welcome all educators to contribute to future editions of the book. We envision introduction of computational culture to the biological education as an ever-expanding and self-organizing process: starting from the second edition, we will work towards unifying the notation and the pedagogical framework based on the students' and instructors' feedback. Meanwhile, the educators have an option of selecting the specific self-contained chapters they like for the courses they teach.

How to use this book?

Since chapters are self-contained, each chapter can be studied or taught individually and chapters can be followed in any order. One can select to cover, for example, a sample of topics from each of the five biological themes in order to obtain a broader view, or cover completely one of the themes for a deeper concentration. Review questions that follow each chapter are helpful to assimilate the material. Additional resources available at the website will be helpful to teachers in preparing their lectures and to students in deeper and broader learning.

The book's website

The book is accompanied by the website www.cambridge.org/b4b containing teaching materials, presentations, and other updates. These can be of help to students as well as educators.

Contributors

The scientists who contributed to this book are leading computational biologists who have ample experience in both research and education. Some are biologists who have became computational over the years, as their computational research needs developed. Others have formal computational background and have made the transition into biology as their research interests and the field developed. All have experienced the need and the difficulty in conveying computational ideas to biology students, and all view this as an important problem that justifies the effort of contributing to this book. They are all committed to the project.

ACKNOWLEDGMENTS

This book would not be possible without the generous support of the Howard Hughes Medical Institute (provided as HHMI award to Pavel Pevzner).

The editors and contributors also thank the editorial team at Cambridge University Press for their continuous and efficient support at all stages of this project. Special thanks go to Megan Waddington, Hans Zauner, Catherine Flack, Lauren Cowles, Zewdi Tsegai, and Katrina Halliday.

Vineet Bafna would like to acknowledge support from the NSF (grant IIS-0810905) and NIH (grant R01 HG004962).

Kun-Mao Chao would like to thank Phillip Compeau, Yao-Ting Huang, and Tandy Warnow for making several valuable comments that improved the presentation. He is supported in part by NSC grants 97-2221-E-002-097-MY3 and 98-2221-E-002-081-MY3 from the National Science Council, Taiwan.

Phillip Compeau and Pavel Pevzner would like to thank Steffen Heber and Glenn Tesler for very helpful comments, as well as Randall Christopher for his superb illustrations.

Mikhail Gelfand is grateful to Mikhail Roytberg, whose approach to the presentation of the dynamic programming algorithm he has borrowed; to Andrey Mironov and Anatoly Rubinov who do not like this approach and have provided very useful comments and critique; to Phillip Compeau for critique and editing (of course, all remaining errors are the author's); and to Pavel Pevzner for the invitation to participate in this volume and patience over failed deadlines.

He acknowledges support from the Ministry of Education and Science of Russia under state contract 2.740.11.0101.

Andrey Grigoriev would like to thank Joe Martin, Chris Lee, and the editorial team for their careful review of his chapter and many helpful suggestions.

Sridhar Hannenhalli would like to acknowledge the support of NIH grant R01GM085226.

Steffen Heber and Brian E. Howard acknowledge the support of many friends and colleagues, who have contributed to their chapter via extremely helpful discussions and feedback. They would especially like to thank Pavel Pevzner, Glenn Tesler, Jens Stoye, Anne Bergeron, and Max Alekseyev. Their work was supported by Education Enhancement Grant (1419) 2008-0273 of the North Carolina Biotechnology Center.

Eugene V. Koonin, Pere Puigbò, and Yuri I. Wolf wish to thank Jian Ma and Pavel Pevzner for many helpful suggestions. Their research is supported through the intramural funds of the US Department of Health and Human Services (National Library of Medicine).

Christopher Lee wishes to thank Pavel Pevzner, Andrey Grigoriev, and the editorial team for their very helpful comments and corrections.

Ran Libeskind-Hadas recognizes that many people have contributed to the content and exposition of this chapter. However, any omissions or errors are entirely his responsibility. Chris Conow, Daniel Fielder, and Yaniv Ovadia wrote the first version of Jane. The version of Jane used in chapter 12, Jane 2.0, is a significant extension of the original Jane software and was designed, developed, and written by Benjamin Cousins, John Peebles, Tselil Schramm, and Anak Yodpinyanee. Professor Catherine McFadden provided valuable feedback on the exposition of the material in this chapter. The development of Jane 2.0 was funded, in part, by the National Science Foundation under grant 0753306 and from the Howard Hughes Medical Institute under grant 52006301. Finally, Professor Michael Charleston inspired the author to work in this field and has been a patient and generous intellectual mentor.

Jian Ma would like to thank Pavel Pevzner, Eugene Koonin, Ryan Cunningham, and Phillip Compeau for helpful suggestions.

Nataša Pržulj thanks Tijana Milenkovic and Wayne Hayes for comments on the chapter.

Russell Schwartz would like to thank Pavel Pevzner, Sridhar Hannenhalli, and Phillip Compeau for helpful comments and discussion. Dr. Schwartz is supported in part by US National Science Foundation award 0612099 and US National Institutes of Health awards 1R01AI076318 and 1R01CA140214. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or National Institutes of Health.

Ron Shamir thanks Hershel Safer for helpful comments, and the support of the Raymond and Beverly Sackler Chair in bioinformatics and of the Israel Science Foundation (grant no. 802/08).

Haixu Tang acknowledges the support of NSF award DBI-0642897.

Tandy Warnow wishes to thank the National Science Foundation for support through grant 0331453; Rahul Suri, Kun-Mao Chao, Phillip Compeau, and Pavel Pevzner for their detailed suggestions that greatly improved the presentation; and Kun-Mao Chao for assistance with making figures for chapter 14.

Tiffani L. Williams and Seung-Jin Sul thank Brian Davis for introducing them to the problem of reconstructing phylogenetic relationships among the big cats. They would also like to thank Danielle Cummings and Suzanne Matthews for their helpful comments on improving this work. Funding for chapter 13 was supported by the National Science Foundation under grants DEB-0629849, IIS-0713618 and IIS-101878.

EDITORS AND CONTRIBUTORS

Editors

Pavel Pevzner
Department of Computer Science and Engineering
University of California at San Diego, USA

Ron Shamir
School of Computer Science
Tel Aviv University, Israel

Contributors

Vineet Bafna
Department of Computer Science and Engineering
University of California at San Diego, USA

Mikhail Gelfand
Department of Bioinformatics and Bioengineering
Moscow State University, Russia

Kun-Mao Chao
Department of Computer Science and Information Engineering
National Taiwan University, Taiwan

Andrey Grigoriev
Department of Biology
Rutgers State University of New Jersey, USA

Phillip Compeau
Department of Mathematics
University of California at San Diego, USA

Sridhar Hannenhalli
Department of Genetics
University of Maryland, USA

Steffen Heber
Department of Computer Science
North Carolina State University, USA

Pere Puigbò
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health, USA

Brian Howard
Department of Computer Science
North Carolina State University, USA

Russell Schwartz
Department of Biological Sciences
Carnegie Mellon University, USA

Eugene Koonin
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health, USA

Seung-Jil Sun
J. Craig Venter Institute
Rockville, USA

Christopher Lee
Department of Chemistry and Biochemistry
University of California at Los Angeles, USA

Haixu Tang
School of Informatics and Computing
Indiana University, USA

Ran Libeskind-Hadas
Department of Computer Science
Harvey Mudd College, USA

Tandy Warnow
Department of Computer Sciences
University of Texas at Austin, USA

Jian Ma
Department of Bioengineering
University of Illinois at Urbana-Champaign, USA

Tiffani Williams
Department of Computer Science and Engineering
Texas A&M University, USA

Nataša Pržulj
Department of Computing
Imperial College London, UK

Yuri Wolf
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health, USA

A COMPUTATIONAL MICRO PRIMER

This introduction is a brief primer on some basic computational concepts that are used throughout the book. The goal is to provide some initial intuition rather than formal definitions. The reader is referred to excellent basic books on algorithms which cover these notions in much greater rigor and depth.

Algorithm

An algorithm is a recipe for carrying out a computational task. For example, every child learns in elementary school how to perform long addition of two natural numbers: "add the right-most digits of the two numbers and write down the sum as the right-most digit of the result. But if the sum is 10 or more, write only the right-most digit and add the leading digit to the sum of the next two digits to the left, etc." We have all learned similar simple procedures for long subtraction, multiplication and division of two numbers. These are all actually simple algorithms. Like any algorithm, each is a procedure that works on inputs (two numbers for the problems above) and produces an output (the result). The same procedure will work on any input, no matter how long it is. While we can carry out simple algorithms on small inputs by hand, computers are needed for more complex algorithms or for longer inputs. As with long addition, a complex task is broken down into simple steps that can be repeated many times, as needed. Algorithms are often displayed for human readers in a short form that summarizes their salient features. One aspect of this simplified representation is that a repeated sequence of steps may be listed only once.

Computational complexity

A basic question in studying algorithms is how efficient they are. For a given input, one can time the computation. Since the time depends on the computer being used, a better understanding of the algorithm can be gained by counting the operations (addition, multiplication, comparison, etc.) performed. This number will be different for different inputs. A common way to evaluate the efficiency of a method is by considering *the number of operations required as a function of the input length.* For example, if an algorithm requires $15n^2$ operations on an input of length n, then we know how many operations will be needed for any input. If we know how many operations our computer performs per second, we can translate this to the running time on our machine.

O notation

Suppose our algorithm requires $15n^2 + 20n + 7$ operations on an n-long input. As n grows larger, the contribution of the lower-order terms $20n + 7$ will become tiny compared to the $15n^2$. In fact, as n grows larger, the constant 15 is not very important when it comes to the *rate* of growth of the number of operations (although it affects the run time).[1] Computer scientists prefer to focus only on the main trend and therefore say that an algorithm that takes $15n^2 + 20n + 7$ operations requires "$O(n^2)$" time (pronounced "oh of n squared"), or, equivalently, is "an $O(n^2)$ algorithm." This means that the algorithm's running time increases quadratically with the input length.[2]

Polynomial and exponential complexity

Some problems can be solved using any of several algorithms, and the O notation is used to decide which algorithm is better (i.e. faster). So an $O(n)$ algorithm is better than an $O(n^2)$ algorithm, which in turn is better than an $O(2^n)$ algorithm. This latter complexity, which is called *exponential* (since n appears in the exponent), is

[1] Computer scientists do not worry too much about the difference between n^2 and $100n^2$, but they greatly worry about the difference between n^3 and $100n^2$. They will typically prefer $100n^2$ to n^3, since for all inputs of length >100 the latter will require more time.
[2] To be precise, "$O(n^2)$" means that the algorithm's run time grows *not more* than quadratically. To specify that the run time is exactly quadratic, complexity theory uses the notation "$\Theta(n^2)$." We shall ignore these differences here.

particularly nasty: as the problem size changes from n to $n + 1$, the run time will double! In contrast, for an O(n) algorithm the run time will grow by O(1), and for an O(n^2) algorithm it will grow by O($2n + 1$). So no matter how fast our computer is, with an algorithm of exponential complexity we shall very quickly run out of computing time as the problem grows: if the problem size grows from 30 to 40, the run time will grow 1024-fold! The main distinction is therefore between *polynomial* algorithms, i.e. those with complexity O(n^c) for some constant c, and exponential ones.

NP-completeness

Computer scientists often try to develop the most efficient algorithm possible for a particular problem. A primary challenge is to find a polynomial algorithm. Many problems do have such algorithms, and then we worry about making the exponent c in O(n^c) as small as possible. For many other problems, however, we do not know of any polynomial algorithm. What can we do when we tackle such a problem in our research? Computer scientists have identified over the years thousands of problems that are not known to be polynomial, and in spite of decades of research currently have only exponential algorithms. On the other hand, so far we do not know how to prove mathematically that they cannot have a polynomial algorithm. However, we know that if any single problem in this set of thousands of problems has a polynomial algorithm, then all of them will have one. So in a sense all these problems are equivalent. We call such problems *NP-complete*. Hence, showing that your problem is NP-complete is a very strong indication that it is hard, and unlikely to have an algorithm that will solve it exactly in polynomial time for every possible input.[3]

Tackling hard problems

So what can one do if the problem is hard? If a problem is NP-complete this means that (as far as we know) it has no algorithm that will solve every instance of the problem exactly in polynomial time. One possible solution is to develop *approximation algorithms*, i.e. algorithms that are polynomial and can approximately solve the problem, by providing (provably) near-optimal but not necessarily always optimal solutions. Another possibility is *probabilistic algorithms*, which solve the

[3] Note that there are problems that were *proven* not to have any polynomial time algorithms, but they are outside the set of established NP-complete problems.

problem in polynomial average time while the worst-case run time can still be exponential. (This would require some assumptions on the probability distribution of the inputs.) Yet another alternative that is often used in bioinformatics is *heuristics* – fast algorithms that aim to provide good solutions in practice, without guaranteeing the optimality or the near-optimality of the solution. Heuristics are typically evaluated on the basis of their performance on the real-life problems they were developed for, without a theoretically proven guarantee for their quality. Finally, *exhaustive algorithms* that essentially try all possible solutions can be developed, and they are often accompanied by a variety of time-saving computational shortcuts. These algorithms typically require exponential time and thus are only practical for modest-sized inputs.