Algorithms on Strings, Trees, and Sequences:
Computer Science and Computational Biology

Dan Gusfield
University of California, Davis

Preface (Abridged)

The history and motivation

Although I didn't know it at the time, I began writing this book in the summer of 1988 when I was part of a computer science research group at the Human Genome Center of Lawrence Berkeley Laboratory. Our group followed the standard assumption that biologically meaningful results could come from considering DNA as a one-dimensional character string, abstracting away the reality of DNA as a flexible three-dimensional molecule, interacting in a dynamic environment with protein and RNA, and repeating a life-cycle in which even the classic linear chromosome exists for only a fraction of the time. A similar, but stronger, assumption existed for protein, holding for example that all the information needed for correct three-dimensional folding is contained in the protein sequence itself, essentially independent of the biological environment the protein lives in. This assumption has recently been modified, but remains largely intact.

For non-biologists, these two assumptions were (and remain) a god-send allowing rapid entry into an exciting and important field. Statements such as

> "The digital information that underlies biochemistry, cell biology, and development can be represented by a simple string of G's, A's, T's and C's. This string is the root data structure of an organism's biology."

reinforced the importance of sequence-level investigation.

So without worrying much about the more difficult chemical and biological aspects of DNA and protein, our computer science group was empowered to consider a variety of biologically important problems defined primarily on sequences, or (more in the computer science vernacular) on strings. We organized our efforts into two high-level tasks. First, to learn the relevant biology, laboratory protocols, and existing algorithmic methods used by biologists. Second to canvass the computer science literature for ideas and algorithms that weren't already used by biologists, but which might plausibly be of use either in current problems, or in problems that we could anticipate arising when vast quantities of sequenced DNA or protein become available.

Our problem

None of us was an expert on string algorithms. At that point I had a textbook knowledge of Knuth-Morris-Pratt, and a deep confusion about Boyer-Moore (under what circumstances it was a linear time algorithm, and how to do strong preprocessing in linear time). I understood the use of dynamic programming to compute edit distance, but otherwise had little exposure to specific string algorithms in biology. My general background was in combinatorial optimization, although I had a prior interest in algorithms for building evolutionary trees and had studied genetics and molecular biology in order to pursue that interest.

41

What we needed then, but didn't have, was a comprehensive cohesive text on string algorithms to guide our education. There were at that time several computer science texts containing a chapter or two on strings, usually devoted to a rigorous treatment of Knuth-Morris-Pratt and a cursory treatment of Boyer-Moore, and possibly an elementary discussion of matching with errors. There were also some good survey papers that had a somewhat wider scope but didn't treat their topics in much depth. There were several texts and edited volumes from the biological side on uses of computers and algorithms for sequence analysis. Some of these were wonderful in exposing the potential benefits and the pitfalls of using computers in biology, but generally lacked algorithmic rigor and covered a narrow range of techniques. Finally, there was the seminal text *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison* edited by D. Sankoff and J. Kruskal, that served as a bridge between algorithms and biology, and had many applications of dynamic programming. But it too was much narrower than our focus, and a bit dated.

Moreover, most of the available sources from either community focused on string matching, the problem of searching for an exact or "nearly exact"- copy of a pattern in a given text. Matching problems are central, but as detailed in this book, they are only a part of the many important computational problems defined on strings. So we recognized that summer a need for a rigorous and fundamental treatment of the general topic of algorithms that operate on strings, along with a rigorous treatment of specific string algorithms of greatest current and potential import in computational biology. This book is an attempt to provide such a dual, and integrated, treatment.

Why mix Computer Science and Computational Biology in one book?

My interest in computational biology began in 1980, when I started reading papers on building evolutionary trees. At that point, computational molecular biology was a largely undiscovered area for computer science, although it was an active area for statisticians and mathematicians (notably Michael Waterman and David Sankoff who have largely framed the field). But seventeen years later, computational biology is hot, and many computer scientists are now entering the (now more hectic, more competitive) field. What should they learn?

The problem is that the emerging field of computational molecular biology is not well defined and its definition is made more difficult by rapid changes in molecular biology itself. Still, algorithms that operate on molecular sequence data (strings) are at the heart of computational molecular biology. The big-picture question in computational molecular biology is how to "do" as much "real biology" as possible by exploiting molecular sequence data (DNA, RNA and protein). Getting sequence data is relatively cheap and fast (and getting more so) compared to more traditional laboratory investigations. The use of sequence data is already central in several subareas of molecular biology and the full impact of having extensive sequence data is yet to be seen. Hence, algorithms that operate on strings will continue to be the area of closest intersection and interaction between computer science and molecular biology. Certainly then, computer scientists need to learn the string techniques that have been most successfully applied. But that is not enough.

Computer scientists need to learn fundamental ideas and techniques that will endure long after today's central motivating applications are forgotten. They need to study methods that prepare them to frame and tackle future problems and applications. Significant contributions to computational biology might be made by extending or adapting algorithms from computer science, even when the original algorithm has no clear utility in biology. Therefore, the computer scientist who wants to enter the general field of computational molecular biology and who learns string algorithms with that end in mind, should receive a training in string algorithms that is much broader than a tour through techniques of known present application. So even if I were to write

a book for computer scientists who only want to do computational biology, I would still choose to include a broad range of algorithmic techniques from pure computer science.

In this book, I cover a wide spectrum of string techniques, well beyond those of established utility, but I select from the many possible illustrations, those techniques that seem to have the greatest potential application in future molecular biology. Potential application, particularly of ideas rather than of concrete methods, and to anticipated rather than to existing problems, is a matter of judgment and speculation. No doubt, some of the material contained in this book will never find direct application in biology, while other material will find uses in surprising ways.

Following the above discussion, this book is a general-purpose rigorous treatment of the entire field of deterministic algorithms that operate on strings and sequences. Many of those algorithms utilize trees as data- structures, or arise in biological problems related to evolutionary trees, hence the inclusion of "trees" in the title.

The model reader is a research-level professional in computer science or a graduate or advanced undergraduate student in computer science, although there are many biologists (and of course mathematicians) with sufficient algorithmic background to read the book. The book is intended to be both a reference, and a main text for courses in pure computer science, and for computer science oriented courses on computational biology.

Explicit discussions of biological applications appear throughout the book, but are more concentrated in the last sections of Part II, and in most of Parts III and IV. I discuss a number of biological issues in detail in order to give the reader a deeper appreciation for the reasons that many biological problems have been cast as problems on strings, and for the variety of (often very imaginative) technical ways that string algorithms have been employed in molecular biology.

This book covers all the classic topics and most of the important advanced techniques in the field of string algorithms, with three exceptions. It only lightly touches on probabilistic analysis, does not discuss parallel algorithms, or the elegant, but very theoretical results on algorithms for infinite alphabets and on algorithms using only constant auxiliary space. The book also does not cover stochastic oriented methods that have come out of the machine learning community, although some of the algorithms in this book are extensively used as subtools in those methods. With these exceptions, the book covers all the major styles of thinking about string algorithms. The reader who absorbs the material in this book will gain a deep and broad understanding of the field, and sufficient sophistication to undertake original research.

Reflecting my background, the book rigorously discusses each of the topics, usually providing complete proofs of behavior (correctness, worst-case time and space). More important, it emphasizes the ideas and derivations of the methods it presents, rather than simply providing an inventory of available algorithms. To better expose ideas and encourage discovery, I often present a complex algorithm by introducing a naive, inefficient, version and then successively applying additional insight and implementation detail to obtain the desired result.

The book contains some new approaches I developed to explain certain classic and complex material. In particular, the preprocessing methods I present for Knuth-Morris-Pratt, Boyer-Moore and several other linear-time pattern matching algorithms, differ from the classical methods, both unifying and simplifying the preprocessing tasks needed for those algorithms. I also expect that my (hopefully simpler and clearer) expositions on linear time suffix tree constructions and on the constant time least common ancestor algorithm will make those important methods more available and widely understood. I connect theoretical results from computer science on sublinear-time algorithms, with widely used methods for biological database search. In the discussion of multiple sequence alignment, I bring together the three major objective functions that have been proposed for multiple alignment, and show a continuity between approximation algorithms for

those three multiple alignment problems. Similarly, the chapter on evolutionary tree construction exposes the commonality of several distinct problems and solutions in a way that is not well known. Throughout the book, I discuss many computational problems concerning repeated substrings (a very widespread phenomenon in DNA). I consider several different ways to define repeated substrings and use each specific definition to explore computational problems and algorithms on repeated substrings.

In the book I try to explain in complete detail, and at a reasonable pace, many complex methods that have previously been written exclusively for the specialist in string algorithms. I avoid detailed code, as I find it rarely serves to explain interesting ideas, and I provide over 400 exercises to both reinforce the material of the book, and to develop additional topics.

Table of Contents

Chapter 4 Semi-Numerical String Matching

PART II Suffix trees and their uses

Chapter 5 Introduction to suffix trees

Chapter 6 Linear time construction of suffix trees

Chapter 7 First applications of suffix trees

Chapter 8 Constant time lowest common ancestor retrieval

Chapter 9 More applications of suffix trees

PART III Inexact Matching, Sequence Alignment, and Dynamic Programming

Chapter 10 The importance of (sub)sequence comparison in molecular biology

Chapter 11 Core string edits, alignments and dynamic programming

Chapter 12 Refining Core String Edits and Alignments

Chapter 13 Extending the core problems

Chapter 14 Multiple String Comparison – The Holy Grail

Chapter 15 Sequence Database and their uses - the MotherLode

PART IV Currents, Cousins and Cameos

Chapter 16 Maps, Mapping, Sequencing and Superstrings

Chapter 17 Strings and Evolutionary Trees

Chapter 18 Three short topics

Chapter 19 Models of genome-level mutations

Chapter 20 Epilogue - Where Next?

Chapter 21 Glossary

Index

Complexity Theory Retrospective II

Lane A. Hemaspaandra and Alan L. Selman, editors

Blurb (in Lieu of the Preface)

Complexity theory is a flourishing area of research that continues to provide one of the richest sources of research problems in computer science. This volume, a collection of articles written by

experts, provides a survey of the subject, a comprehensive guide to research, and a provocative look to the future.

The editors' aim has been to provide an accessible description of the current state of complexity theory and to demonstrate the breadth of techniques and results that make the subject exciting. Papers are on traditional topics ranging from sublogarithmic space to exponential time, on new combinatorial techniques and recent successes such as interactive proof systems, and on the newly emerging areas of quantum and biological computing. As a result, researchers and students in computer science will find this book an excellent starting point for study of the subject and a useful source of the key known results.

Table of Contents

Models of Computation
Exploring the Power of Computing

John E. Savage
Brown University

Preface (Abridged)

Theoretical computer science treats any computational subject for which a good model can be created. Research on formal models of computation was initiated in the 1930s and 1940s by Turing, Post, Kleene, Church, and others. In the 1950s and 1960s programming languages, language translators, and operating systems were under development and therefore became the subject and

basis for a great deal of theoretical work. The power of computers of this period was limited by slow processors and small amounts of memory, and thus theories (models, algorithms, and analysis) were developed to explore the efficient use of computers as well as the inherent complexity of problems. The former subject is known today as algorithms and data structures, the latter computational complexity.

The focus of theoretical computer scientists in the 1960s on languages is reflected in the first textbook on the subject, *Formal Languages and Their Relation to Automata* by John Hopcroft and Jeffrey Ullman. This influential book led to the creation of many language-centered theoretical computer science sources; many introductory theory courses today continue to reflect the content of this book and the interests of theoreticians of the 1960s and early 1970s.

Although the 1970s and 1980s saw the development of models and methods of analysis directed at understanding the limits on the performance of computers, this attractive new material has not been made available at the introductory level. This book is designed to remedy this situation.

This book is distinguished from others on theoretical computer science by its primary focus on real problems, its emphasis on concrete models of machines and programming styles, and the number and variety of models and styles it covers. These include the logic circuit, the finite state machine, the pushdown automaton, the random-access machine, memory hierarchies, the PRAM (parallel random-access machine), the VLSI (very large-scale integrated) chip, and a variety of parallel machines.

The book covers the traditional topics of formal languages and automata and complexity classes but also gives an introduction to the more modern topics of space-time tradeoffs, memory hierarchies, parallel computation, the VLSI model, and circuit complexity. These modern topics are integrated throughout the text as illustrated by the early introduction of P-complete and NP-complete problems. The book provides the first textbook treatment of space-time tradeoffs and memory hierarchies as well as a comprehensive introduction to traditional computational complexity. Its treatment of circuit complexity is modern and substantive, and parallelism is integrated throughout.

Table of Contents

54

9 Circuit Complexity   391

10 Space-Time Tradeoffs   461

11 Memory-Hierarchy Tradeoffs   529

12 VLSI Models of Computation   575

The Theory of Computation

Bernard M. Moret
University of New Mexico

Preface

Theoretical computer science covers a wide range of topics, but none is as fundamental and as useful as the theory of computation. Given that computing is our field of endeavor, the most basic question that we can ask is surely "What can be achieved through computing?``"

In order to answer such a question, we must begin by defining computation, a task that was started last century by mathematicians and remains very much a work in progress at this date. Most theoreticians would at least agree that computation means solving problems through the mechanical, preprogrammed execution of a series of small, unambiguous steps. From basic philosophical ideas about computing, we must progress to the definition of a model of computation, formalizing these basic ideas and providing a framework in which to reason about computation. The model must be a framework in which to reason about computation. The model must be both reasonably realistic (it cannot depart too far from what is perceived as a computer nowadays) and as universal and powerful as possible. With a reasonable model in hand, we may proceed to posing and resolving fundamental questions such as "What can and cannot be computed?``" and "How efficiently can something be computed?`"" The first question is at the heart of the theory of computability and the second is at the heart of the theory of complexity.

In this text, I have chosen to give pride of place to the theory of complexity. My basic reason is very simple: complexity is what really defines the limits of computation. Computability establishes some absolute limits, but limits that do not take into account any resource usage are hardly limits in a practical sense. Many of today`s important practical questions in computing are based on resource problems. For instance, encryption of transactions for transmissions over a network can never be entirely proof against snoopers, because an encrypted transaction must be decrypted by some means and thus can always be deciphered by someone determined to do so, given sufficient resources. However, the real goal of encryption is to make it sufficiently "hard``"—that is, sufficiently resource-intensive—to decipher the message that snoopers will be discouraged or that even determined spies will take too long to complete the decryption. In other words, a good encryption scheme does not make it impossible to decode the message, just very difficult—the problem is not one of computability but one of complexity. As another example, many tasks carried out by computers today involve some type of optimization: routing of planes in the sky or of packets through a network so as to get planes or packets to their destination as efficiently as possible; allocation of manufactured products to warehouses in a retail chain so as to minimize waste and further shipping; processing of raw materials into component parts (e.g., cutting cloth into patterns pieces or cracking crude oil into a range of oils and distillates) so as to minimize waste; designing new products to minimize production costs for a given level of performance; and so forth. All of these problems are certainly computable: that is, each such problem has a well-defined optimal solution that could be found through sufficient computation (even if this computation is nothing more than an exhaustive search through all possible solutions). Yet these problems are so complex that they cannot be solved optimally within a reasonable amount of time; indeed, even deriving good approximate solutions for these problems remains resource-intensive. Thus the complexity of solving (exactly

or approximately) problems is what determines the usefulness of computation in practice. It is no accident that complexity theory is the most active area of research in theoretical computer science today.

Yet this text is not just a text on the theory of complexity. I have two reasons for covering additional material: one is to provide a graduated approach to the often challenging results of complexity theory and the other is to paint a suitable backdrop for the unfolding of these results. The backdrop is mostly computability theory—clearly, there is little use in asking what is the complexity of a problem that cannot be solved at all! The graduated approach is provided by a review chapter and a chapter on finite automata. Finite automata should already be somewhat familiar to the reader; they provide an ideal testing ground for the ideas and methods need in working with complexity models. On the other hand, I have deliberately omitted theoretical topics (such as formal grammars, the Chomsky hierarchy, formal semantics, and formal specifications) that, while interesting in their own right, have limited impact on everyday computing—some because they are not concerned with resources, some because the models used are not well accepted, and grammars because their use in compilers is quite different from their theoretical expression in the Chomsky hierarchy. Finite automata and regular expressions (the lowest level of the Chomsky hierarchy) are covered here but only by way of an introduction to (and contrast with) the universal models of computation used in computability and complexity.

Of course, not all results in the theory of complexity have the same impact on computing. Like any rich body of theory, complexity theory has applied aspects and very abstract ones. I have focused on the applied aspects: for instance, I devote an entire chapter on how to prove that a problem is hard but less than a section on the entire topic of structure theory (the part of complexity theory that addresses the internal logic of the field). Abstract results found in this text are mostly in support of fundamental results that are later exploited for practical reasons.

Since theoretical computer science is often the most challenging topic studied in the course of a degree program in computing, I have avoided the dense presentation often favored by theoreticians (definitions, theorems, proofs, with as little text in between as possible). Instead, I provide intuitive as well as formal support for further derivations and present the idea behind any line of reasoning before formalizing said reasoning. I have included large numbers of examples and illustrated many abstract ideas through diagrams; the reader will also find useful synopses of methods (such as steps in an NP-completeness proof) for quick reference. Moreover, this text offers strong support through the Web for both students and instructors. Instructors will find solutions for most of the 250 problems in the text, along with many more solved problems; students will find interactive solutions for chosen problems, testing and validating their reasoning process along the way rather than delivering a complete solution at once. In addition, I will also accumulate on the Web site addenda, errata, comments form students and instructors, and pointers to useful resources, as well as feedback mechanisms—I want to hear from all users of this text suggestions on how to improve it. The URL for the Website is `http://www.cs.unm.edu/~moret/computation/`; my email address is `moret@cs.unm.edu`.

Table of Contents