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

Dan Gusfield
University of California, Davis

Cambridge University Press, 1998
ISBN: 0-521-58519-8

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.

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

Cole's proof / 3.2.2 The case when the pattern does occur in the text / 3.2.3 Adding in the bad character rule / 3.3 The original preprocessing for Knuth-Morris-Pratt / 3.3.1 The method does not use fundamental preprocessing / 3.3.2 The easy case / 3.3.3 The general case / 3.3.4 The complete preprocessing algorithm / 3.3.5 How to compute the optimized shift values / 3.4 Exact matching with a set of patterns / 3.4.1 Naive use of keyword trees for set matching / 3.4.2 The speed-up: generalizing Knuth-Morris-Pratt / 3.4.3 Failure functions for the keyword tree / 3.4.4 The failure links speed up the search / 3.4.5 Linear preprocessing for the failure function / 3.4.6 The full Aho-Corasick algorithm: relaxing the substring assumption / 3.4.6.1 Implementation / 3.5 Three applications of exact set matching / 3.5.1 Matching against a DNA or protein library of known patterns / 3.5.2 Exact matching with wild cards / 3.5.2.1 Correctness and complexity of the method / 3.5.3 Two dimensional exact matching / 3.6 Regular Expression Pattern Matching / 3.6.1 Formal Definitions / 3.7 Exercises

Chapter 4 Semi-Numerical String Matching

4.1 Arithmetic verses comparisons / 4.2 The Shift-And method / 4.2.1 How to construct array M / 4.2.2 Shift-And is effective for small patterns / 4.2.3 agrep: the Shift-And method with errors / 4.2.4 How to compute Mk / 4.3 The match-count problem and Fast Fourier Transform / 4.3.1 A fast worst-case method for the match-count problem? / 4.3.2 Using Fast Fourier Transform for match-counts / 4.3.2.1 The high-level approach / 4.3.2.2 Cyclic correlation / 4.3.2.3 Handling wildcards in match-counts / 4.4 Karp-Rabin fingerprint methods for exact match / 4.4.1 Arithmetic replaces comparisons / 4.4.2 Fingerprints of P and T / 4.4.2.1 Prime moduli limit false matches / 4.4.2.2 The Central Theorem / 4.4.2.3 Extensions / 4.4.2.4 Even lower limits on error / 4.4.2.5 Checking for error in linear time / 4.4.3 Why fingerprints? / 4.5 Exercises

PART II Suffix trees and their uses

Chapter 5 Introduction to suffix trees

5.1 A short history / 5.2 Basic definitions / 5.3 A motivating example / 5.4 A naive algorithm to build a suffix tree

Chapter 6 Linear time construction of suffix trees

6.1 Ukkonen's linear time suffix tree algorithm / 6.1.1 Implicit suffix trees / 6.1.2 Ukkonen's algorithm at a high level 6.1.3 Implementation and speedup / 6.1.3.1 Suffix links: first implementation speedup / 6.1.3.2 Following a trail of suffix links to build I(i+1) / 6.1.3.3 Single extension algorithm: SEA / 6.1.3.4 Trick number 1: skip/count trick / 6.1.4 A simple implementation detail / 6.1.5 Two more little tricks and we're done / 6.1.5.1 The punch line / 6.1.6 Creating the true suffix tree / 6.2 Weiner's linear time suffix tree algorithm / 6.2.1 A straightforward construction / 6.2.2 Towards a more efficient implementation / 6.2.2.1 Finding Head(i) efficiently / 6.2.3 The basic idea of Weiner's algorithm / 6.2.3.1 The algorithm in the good case / 6.2.3.2 The two degenerate cases / 6.2.4 The Full Algorithm for creating T(i) from T(i+1) / 6.2.4.1 Correctness / 6.2.4.2 How to update the vectors / 6.2.5 Time analysis of Weiner's algorithm / 6.2.6 Last comments about Weiner's algorithm / 6.3 McCreight's suffix tree

algorithm / 6.4 Generalized suffix tree for a set of strings / 6.5 Practical implementation issues / 6.5.1 Alphabet independence: all linears are equal, but some are more equal than others / 6.6 Exercises

Chapter 7 First applications of suffix trees

7.1 APL1: Exact string matching / 7.2 APL2: Suffix trees and the exact set matching problem / 7.2.1 Comparing suffix trees and keyword trees for exact set matching / 7.3 APL3: The substring problem for a database of patterns / 7.4 APL4: Longest common substring of two strings / 7.5 APL5: Recognizing DNA contamination / 7.6 APL6: Common substrings of more than two strings / 7.6.1 Computing the C(v) numbers / 7.7 APL7: Building a smaller directed graph for exact matching / 7.8 APL8: A reverse role for suffix trees, and major space reduction / 7.8.1 Matching statistics: duplicating bounds and reducing space / 7.8.2 Correctness and time analysis for matching statistics / 7.8.3 A small but important extension / 7.9 APL9: Space efficient longest common substring / 7.10 APL10: All-Pairs Suffix-Prefix Matching / 7.10.1 Solving the all-pairs suffix-prefix problem in linear time / 7.11 Introduction to repetitive structures in strings / 7.11.1 Repetitive structures in biological strings / 7.11.2 Uses of repetitive structures in molecular biology / 7.12 APL11: Finding all maximal repetitive structures in linear time / 7.12.1 A linear time algorithm to find all maximal repeats / 7.12.2 Finding supermaximal repeats in linear time / 7.12.3 Finding all the maximal pairs in linear time / 7.13 APL12: Circular string linearization / 7.13.1 Solution via suffix trees / 7.14 APL13: Suffix Arrays: more space reduction / 7.14.1 Suffix tree to suffix array in linear time / 7.14.2 How to search for a pattern using a suffix array / 7.14.3 A simple accelerant / 7.14.4 A super-accelerant / 7.14.5 How to obtain the Lcp values / 7.14.6 Where do large alphabet problems arise? / 7.15 APL14: Suffix trees in genome-scale projects / 7.16 APL15: A Boyer-Moore approach to exact set matching / 7.16.1 The search / 7.16.2 Bad character rule / 7.16.3 Good suffix rule / 7.16.4 How to determine i2 and i3 / 7.16.5 An implementation eliminating redundancy / 7.17 APL16: Ziv-Lempel data compression / 7.17.1 Implementation using suffix trees / 7.17.2 A one-pass version / 7.17.3 The real Ziv-Lempel / 7.18 APL17: Minimum length encoding of DNA / 7.19 Additional applications / 7.20 Exercises

Chapter 8 Constant time lowest common ancestor retrieval

8.1 Introduction / 8.1.1 What do ancestors have to do with strings? / 8.2 The assumed machine model / 8.3 Complete binary trees: A very simple case / 8.4 How to solve lca queries in B / 8.5 First steps in mapping T to B / 8.6 The mapping of T to B / 8.7 The linear time preprocessing of T / 8.8 Answering an lca query in constant time / 8.9 The binary tree is only conceptual / 8.10 For the purists: how to avoid bit-level operations / 8.11 Exercises

Chapter 9 More applications of suffix trees

9.1 Longest common extension: a bridge to inexact matching / 9.1.1 Linear time solution / 9.1.2 Space efficient longest common extension / 9.2 Finding all maximal palindromes in linear time / 9.2.1 Linear time solution / 9.2.2 Complemented and

separated palindromes / 9.3 Exact matching with wild cards / 9.4 The k-mismatch problem / 9.4.1 The solution / 9.5 Approximate palindromes and repeats / 9.6 Faster methods for tandem repeats / 9.6.1 The speedup for k-mismatch tandem repeats / 9.7 A linear time solution to the multiple common substring problem / 9.7.1 The method / 9.7.2 Time analysis / 9.7.3 Related uses / 9.8 Exercises

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

11.1 Introduction / 11.2 The edit distance between two strings / 11.2.1 String alignment / 11.3 Dynamic programming calculation of edit distance / 11.3.1 The recurrence relation / 11.3.2 Tabular computation of edit distance / 11.3.3 The traceback / 11.3.3.1 The pointers represent all optimal edit transcripts / 11.4 Edit graphs / 11.5 Weighted edit distance / 11.5.1 Operation weights / 11.5.2 Alphabet-weight edit distance / 11.6 String similarity / 11.6.1 Computing similarity / 11.6.2 Special cases of similarity / 11.6.3 Alignment graphs for similarity / 11.6.4 Endspace free variant / 11.6.5 Approximate occurrences of P in T / 11.7 Local alignment: Finding substrings of high similarity / 11.7.1 Computing local alignment / 11.7.2 How to solve the local suffix alignment problem / 11.7.3 Three final comments on local alignment / 11.8 Gaps / 11.8.1 Introduction to Gaps / 11.8.2 Why gaps? / 11.8.3 cDNA matching: a concrete illustration / 11.8.3.1 Processed Pseudogenes / 11.8.4 Choices for gap weights / 11.8.5 Arbitrary gap weights / 11.8.5.1 Time analysis / 11.8.6 Affine (and constant) Gap Weights / 11.8.6.1 The recurrences / 11.8.6.2 Time analysis / 11.9 Exercises

Chapter 12 Refining Core String Edits and Alignments

12.1 Computing alignments in only linear space / 12.1.1 Space reduction for computing similarity / 12.1.2 How to find the optimal alignment in linear space / 12.1.3 The full idea: Use recursion / 12.1.4 Time analysis / 12.1.5 Extension to local alignment / 12.2 Faster algorithms when the number of differences is bounded / 12.2.1 Where do bounded difference problems arise? / 12.2.2 Illustrations from molecular biology / 12.2.3 k-difference global alignment / 12.2.4 The return of the suffix tree: k-difference inexact matching / 12.2.5 The primer (and probe) selection problem revisited / 12.2.5.1 How to solve the k-difference primer problem / 12.3 Exclusion methods: fast expected running time / 12.3.1 The BYP method / 12.3.2 Expected time analysis of Algorithm BYP / 12.3.3 The Chang-Lawler method / 12.3.4 Multiple filtration for k-mismatches / 12.3.5 Myers' sublinear-time method / 12.3.6 Final comment on exclusion methods / 12.4 Yet more suffix-trees, more hybrid dynamic programming / 12.4.1 The P-against-all problem / 12.4.2 The (threshold) all-against-all problem / 12.4.2.1 Correctness and time analysis / 12.5 A faster (combinatorial) algorithm for longest common subsequence / 12.5.1 Longest increasing subsequence / 12.5.2 Longest common subsequence reduces to longest increasing sequence / 12.5.3 How good is the method / 12.5.4 The lcs of more than two strings / 12.6 Convex Gap weights / 12.6.1 Forward dynamic programming / 12.6.2 The basis of the speedup / 12.6.3 Cell pointers and row partition / 12.6.3.1

Preparation for the speedup / 12.6.4 Final implementation details and time analysis / 12.6.4.1 The case of F values is essentially symmetric / 12.7 The Four-Russians speedup / 12.7.1 t-blocks / 12.7.2 The Four-Russians idea for the restricted block function / 12.7.2.1 Accounting detail / 12.7.3 The trick: offset encoding / 12.7.3.1 Time analysis / 12.7.4 Practical approaches / 12.8 Exercises

Chapter 13 Extending the core problems

13.1 Parametric Sequence Alignment / 13.1.1 Introduction / 13.1.2 Definitions and first results / 13.1.3 Parametric alignment with the use of scoring matrices / 13.1.4 Efficient algorithms for computing a polygonal decomposition / 13.1.4.1 Finding a polygon of the decomposition / 13.1.4.2 Filling in the parameter space / 13.1.5 Time analysis and next idea / 13.1.6 Bounding the number of polygons in the decomposition / 13.1.6.1 The special case of global alignment / 13.1.7 Uses for parametric alignment / 13.1.7.1 Sensitivity analysis / 13.1.7.2 Efficient computation of all co-optimals / 13.2 Computing suboptimal alignments / 13.2.1 First definitions and first results / 13.2.2 A useful re-weighting / 13.2.3 Counting and enumerating near-optimal paths / 13.2.4 An alternative approach to suboptimal alignment / 13.3 Chaining diverse local alignments / 13.4 Exercises

Chapter 14 Multiple String Comparison – The Holy Grail

14.1 Why multiple string comparison? / 14.1.1 Biological basis for multiple string comparison / 14.2 Three "big-picture" biological uses for multiple string comparison / 14.3 Family and superfamily representation / 14.3.1 Family representations and alignments with profiles / 14.3.1.1 Aligning a string to a profile / 14.3.2 Signature representations of families / 14.3.2.1 Signatures for Helicase proteins / 14.4 Multiple sequence comparison for structural inference / 14.5 Introduction to computing multiple string alignments / 14.5.1 How to score multiple alignments / 14.6 Multiple alignment with the sum-of-pairs (SP) objective function / 14.6.1 An exact solution to the SP alignment problem / 14.6.1.1 A speed up for the exact solution / 14.6.2 A bounded-error approximation method for SP alignment / 14.6.2.1 An initial key idea: Alignments consistent with a tree / 14.6.2.2 The center star method for SP alignment / 14.6.3 Weighted SP alignment / 14.7 Multiple alignment with consensus objective functions / 14.7.1 Steiner consensus strings / 14.7.2 Consensus strings from multiple alignment / 14.7.3 Approximating the optimal consensus multiple alignment / 14.8 Multiple alignment to a (phylogenetic) tree / 14.8.1 A heuristic for phylogenetic alignment / 14.8.1.1 The error analysis / 14.8.1.2 Computing the minimum distance lifted alignment / 14.9 Comments on bounded-error approximations / 14.10 Common multiple alignment methods / 14.10.1 Iterative pairwise alignment / 14.10.2 Two specific illustrations of iterative pairwise alignment / 14.10.2.1 Iterative multiple alignment to identify protein secondary structure / 14.10.2.2 Iterative multiple alignment to build evolutionary trees / 14.10.3 Repeated-motif methods / 14.10.4 Two newer approaches to multiple string comparison / 14.11 Exercises

Chapter 15 Sequence Database and their uses - the MotherLode

15.1 Success stories of database search / 15.1.1 The first success story / 15.1.2 A more recent example of successful database search / 15.1.3 Indirect applications of database search / 15.2 The database industry / 15.3 Algorithmic issues in database search / 15.3.1 Should there be any? / 15.4 Real Sequence Database search / 15.5 FASTA / 15.6 BLAST / 15.6.1 The hit (hot-spot) strategy of BLAST / 15.6.2 The effectiveness of BLAST / 15.7 PAM: the first major amino acid substitution matrices / 15.7.1 PAM units and PAM matrices / 15.7.2 PAM units / 15.7.3 PAM matrices / 15.7.4 How are PAM matrices actually derived? / 15.7.5 The use of the PAM matrix / 15.8 PROSITE / 15.9 BLOCKS and BLOSUM / 15.10 The BLOSUM substitution matrices / 15.11 Additional considerations for database searching / 15.11.1 Statistical significance / 15.11.2 A theory of log-odds scores / 15.11.3 Importance of searching protein with protein / 15.12 Exercises

PART IV Currents, Cousins and Cameos

Chapter 16 Maps, Mapping, Sequencing and Superstrings

16.1 A look at some DNA mapping and sequencing problems / 16.2 Mapping and the genome project / 16.3 Physical versus genetic maps / 16.4 Physical mapping / 16.5 Physical mapping: STS-content mapping and ordered clone libraries / 16.5.1 Reconstruction of STS order / 16.6 Physical mapping: Radiation-hybrid mapping / 16.6.1 Reconstruction of STS order in radiation hybrids / 16.6.2 Traveling salesman formulation of STS ordering / 16.6.3 Back to STS-content mapping: the case of errors / 16.7 Physical mapping: Fingerprinting for general map construction / 16.8 Computing the tightest layout / 16.9 Physical mapping: last comments / 16.10 An introduction to map alignment / 16.10.1 A non-unary dynamic programming approach to map alignment / 16.10.2 Extensions of the map alignment model / 16.11 Large-scale sequencing and sequence assembly / 16.12 Directed sequencing / 16.13 Top-down, bottom-up sequencing: The picture using YACs / 16.13.1 Is mapping necessary for sequencing? / 16.13.2 Fragment selection for sequencing / 16.13.3 Some real numbers / 16.14 Shotgun DNA sequencing / 16.15 Sequence assembly / 16.15.1 Step one: Overlap detection / 16.15.2 Step two: string layout / 16.15.3 Step three: Deciding the consensus / 16.16 Final comments on top-down, bottom-up sequencing / 16.17 The shortest superstring problem / 16.17.1 Basic Definitions / 16.17.2 The objective function for superstrings / 16.17.3 Cyclic strings and cycle covers / 16.17.4 How cycle covers define superstrings / 16.17.5 Factor-of-four approximation / 16.17.5.1Error analysis of the algorithm / 16.17.6 Improvement to a factor of three / 16.17.6.1 Error analysis / 16.17.7 Efficient implementation / 16.17.7.1 Non-trivial cycle cover / 16.16.7.2 How to form the matrix efficiently / 16.18 Sequencing by hybridization / 16.18.1 Reduction to Euler paths / 16.18.2 Continuity of compatible strings / 16.18.3 Last comments on SBH / 16.19 Exercises

Chapter 17 Strings and Evolutionary Trees

17.1 Ultrametric trees and ultrametric distances / 17.1.1 Introduction / 17.1.2 Evolutionary trees as ultrametric trees / 17.1.3 How to test for an ultrametric tree / 17.1.4 How are ultrametric data obtained? / 17.1.4.1 Laboratory-based methods / 17.1.4.2 Sequence-based methods / 17.1.4.3 Final comments / 17.2 Additive-distance trees /

17.2.1 Introduction / 17.2.2 Algorithms for the additive tree problem / 17.2.2.1 Compact additive trees / 17.3 Parsimony: character-based evolutionary reconstruction / 17.3.1 Introduction / 17.3.2 Where do character data come from? / 17.3.3 Perfect Phylogeny / 17.3.4 An O(nm)-time algorithm for the perfect phylogeny problem / 17.3.5 Tree compatibility: an application of perfect phylogeny / 17.3.6 Generalized perfect phylogeny / 17.4 The centrality of the ultrametric problem / 17.4.1 The additive tree problem viewed as an ultrametric problem / 17.4.2 The perfect phylogeny problem viewed as an ultrametric problem / 17.5 Maximum parsimony, Steiner trees and Perfect Phylogeny / 17.5.1 Basic definitions / 17.5.2 Approximations to maximum parsimony / 17.6 Phylogenetic Alignment, again / 17.6.1 The Fitch-Hartigan minimum mutation problem / 17.6.2 Phylogenetic alignment used to compute PAM matrices / 17.7 Connections between multiple alignment and tree construction / 17.8 Exercises

Chapter 18 Three short topics

18.1 Matching DNA to protein with frameshift errors / 18.1.1 Matching a string to a network / 18.1.2 DNA/protein matching cast as network matching / 18.2 Gene prediction / 18.2.1 Exon assembly / 18.3 Molecular computation: Computing with (not about) DNA strings / 18.3.1 Lipton's approach to the Satisfiability Problem / 18.3.2 Critique / 18.4 Exercises

Chapter 19 Models of genome-level mutations

19.1 Introduction / 19.1.1 Genome rearrangements give new evolutionary insights / 19.2 Genome rearrangements with inversions / 19.2.1 Definitions and initial facts / 19.2.2 The heuristics / 19.2.2.1 Improving the guarantee / 19.3 Signed inversions / 19.4 Exercises

Chapter 20 Epilogue - Where Next?

Chapter 21 Glossary

Index

Complexity Theory Retrospective II

Lane A. Hemaspaandra and Alan L. Selman, editors

Springer-Verlag, New York, 1997
ISBN 0-387-94973-9
http://www.springer-ny.com/catalog/np/mar97np/DATA/0-387-94973-9.html

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

5 A Taxonomy of Proof Systems — Oded Goldreich

5.1 Introduction / 5.2 A Technical Exposition / 5.2.1 Interactive Proof Systems / 5.2.2 MIP and PCP / 5.2.3 Computationally Sound Proof Systems / 5.2.4 Other Types of Proof Systems / 5.2.5 Comparison / 5.3 The Story / 5.3.1 The Evolution of Proof Systems / 5.3.2 PCP and Approximation / 5.3.3 Interactive Proofs and Program Checking / 5.3.4 Zero-Knowledge Proofs

6 Structural Properties of Complete Problems for Exponential Time — Steven Homer

6.1 Introduction / 6.2 Strong Reductions to Complete Sets / 6.3 Immunity for Complete Problems / 6.4 Differences between Complete Sets / 6.5 Other Properties and Open Problems / 6.5.1 Properties of "Weak" Complete Sets / 6.5.2 Polynomial-Time Complete Recursively Enumerable Sets / 6.5.3 A Short List of Open Problems

7 The Complexity of Obtaining Solutions for Problems in NP and NL — Birgit Jenner and Jacobo Toran

7.1 Introduction / 7.2 Computing Optimal Solutions: The Class FP^{NP} / 7.3 Bounded Queries to NP / 7.4 Computing Solutions Uniquely: The Class NPSV / 7.5 Nonadaptive Queries to NP: The Class $\mathrm{FP}_t t^{N}P$ / 7.6 A Look inside Nondeterministic Logspace / 7.7 Conclusions

8 Biological Computing — Stuart A. Kurtz, Stephen R. Mahaney, James S. Royer, and Janos Simon

8.1 Introduction / 8.2 The One-Molecule Processor / 8.3 A Brief Introduction to Biochemistry / 8.3.1 DNA, RNA, and Proteins / 8.3.2 Protein Synthesis / 8.4 Computational Molecules / 8.4.1 CNA / 8.4.2 tCNA / 8.4.3 The Synthesis of tCNA / 8.5 The Microarchitecture of CNA Computers / 8.6 A Brief Discussion of Adleman's Model Versus Our Model / 8.7 Conclusions

9 Computing with Sublogarithmic Space — Maciej Liskiewicz and Ruediger Reischuk

9.1 Are Sublogarithmic Space Classes of Any Interest? / 9.2 The Alternating Sublogarithmic Space World / 9.3 Adding Randomness / 9.4 Special Limitations of Machines with a Sublogarithmic Space Bound / 9.4.1 Technical Preliminaries / 9.4.2 Inputs with a Periodic Structure / 9.4.3 Fooling ATMs / 9.5 A Survey of Lower Space Bound Proofs / 9.5.1 Languages for Separating the Levels of the Alternation Hierarchy / 9.5.2 ATMs with a Constant Number of Alternations / 9.5.3 Unbounded Alternation / 9.5.4 Closure Properties / 9.5.5 Lower Bounds for Context-Free Languages / 9.6 Conclusions and Open Problems

10 The Quantitative Structure of Exponential Time — Jack H. Lutz

Models of Computation

Exploring the Power of Computing

John E. Savage

Brown University

Addison Wesley Longman, 1998

ISBN: 0-201-89539-0

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

The Theory of Computation

Bernard M. Moret
University of New Mexico

Addison Wesley Longman, 1998
ISBN 0-201-25828-5

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