The Book Review Column[1]

by William Gasarch
Department of Computer Science
University of Maryland at College Park
College Park, MD, 20742
email: gasarch@cs.umd.edu

In this column we review the following books.

1. **Foundations of Logic and Mathematics: Applications to Computer Science and Cryptography** by Yves Nievergelt. Review by Saif Terai. This book is in two parts: Theory and Applications. The Theory is Logic, the applications are cryptography.

2. **Rippling: Meta-Level Guidance For Mathematical Reasoning** by Alan Bundy, David Basin, Dieter Hutter, and Andrew Ireland. Review by Maulik Dave. The book is about Rippling, which is a proof planning technique. Automatic theorem proving is done by rewrite rules. The conjecture, assumptions, and already proved theorems are supplied to an automatic theorem prover in the form of rewrite rules.

3. **Handbook of Nature-Inspired and Innovative Computing** Edited by Albert Y. Zomaya. Review by Aaron Sterling. This books goal is to be a Virtual Get Together of several researchers that one could invite to attend a conference on *futurism* dealing with the theme of Computing in the 21st Century.

4. **Algorithms and Data Structures: The Basic Toolbox** by Kurt Mehlhorn and Peter Sanders. Review by Hal C. Elrod. What are the basic tools of the accomplished programmer? Read the book to find out.

5. **The Algorithm Design Manual (Second Edition)** By Steven S. Skiena. Reviewed by Neelakantan Kartha. This is an algorithms book for the practical person who actually wants to implement stuff.

6. **Graph Theory: A Problem Oriented Approach** by Daniel Marcus. Review by Haris Aziz. This is a textbook where the reader is given some relevant information but must then do problems, problems, and more problems.

7. **Proofs from THE BOOK (4th edition)** by Martin Aigner and Günter M. Ziegler. Review by Miklós Bóna. Paul Erdős would say that there is a book somewhere, possibly in heaven, and that book contained the nicest and most elucidating proof of every theorem in mathematics. Hence the term **Proofs from THE BOOK** has come to mean particularly pleasing proofs. This book is a collection of such.

8. **Handbook of Chemoinformatics Algorithms** Edited by Faulon, Bender. Review by Aaron Sterling. What are the most common chemoinformatics algorithms? What is a chemoinformatics algorithms? Read the review to find out.

9. **Dynamic Fuzzy Logic and its Applications** by Fanzhang. Review by Song Yan. In Fuzzy logic, rather than an element being in or out of a set it is given a value between 0 and 1 to indicate how much it is in. What if we allow those values to change over time? You obtain Fuzzy Logic.

BOOKS I NEED REVIEWED FOR SIGACT NEWS COLUMN

1. *Introduction to Bio-ontologies* by Robinson and Bauer.
2. *The Dots and Boxes Game: Sophisticated Child's play* By Berlekamp.
3. *New Mathematical Diversions* by Martin Gardner.
4. *The Magic Numbers of the Professor* by O'Shea and Dudley.

Review of[2] of
Foundations of Logic and Mathematics
Applications to Computer Science and Cryptography
by Yves Nievergelt
Birkhauser, Springer, 2002
430 pages

Review by Saif Terai terais@algonquincollege.com

1 Overview

This book could serve as a text and as a reference; it has a different presentation style. The author states, for example, a theorem, a rule, a remark or a definition, followed by an example. Examples are an important part of the presentation and should be read for continuity; they are an important component of understanding. The opening paragraph on the back cover arouses the readers interest, with questions such as, "Why is the truth table for logical implication so unintuitive? Why are there no recipes to design proofs? Where do these numerous mathematical rules come from? What issues in logic, mathematics and computer science still remain unresolved? In what ways are we going to use this material?" The book is divided in two parts; Part A is titled Theory and Part B Applications. It has a five and a half page bibliography, and a 13 page extensive index. I checked with the publisher for errata and solutions, none are available for the book. Some symbols used in the book are not listed in the index page. Most symbols in the index page have a description, but some do not.

2 Summary of Contents

Chapter 0 - Boolean Algebraic Logic An introduction to this chapter explains the use of mathematics and logical reasoning used for several millennia. The need for logic, mathematics and computing is justified. Its use can be traced to ancient civilizations in Babylonia, China and India. Its use includes calculation of taxes, dates of astronomical events such as eclipses and change of seasons. Terms of logic are introduced, such as hypothesis, conclusion, implication, converse and contrapositive The concept of an empty set is defined. A table lists the alphabets for logic and set theory. For example the symbols for empty set, "belongs to", "equals", "for each", "there exists" are listed. The section on well formed formulae defines an atomic formula; infix, prefix and postfix notation are differentiated. Other sections in the chapter include tautologies and contradictions, proofs by tautologies, proofs by contradictions, synthesis of logical formulae and synthesis by Karnaugh's tables.

Chapter 1 - Logic and Deductive Reasoning A 42 page chapter begins with a good introduction on how logic is used in reasoning. It explains how features in natural phenomenon that cannot be seen or perceived through the senses do indeed exist; (other) observations and deductive reasoning lead us to conclusions about those hidden features. The shape of orbits are discussed; statements that we widely considered true for thousands of years have become false when new evidence was

uncovered. Rules of inference, axioms, substitutions and Modus Ponens, minor and major premise are defined with examples. This section ends with Derived Rule and Reflexive Law of Implication. Section 1.2 covers Classical and Implicational Calculus. Implicational calculus is a version of Propositional Calculus. It uses only one connective called implicational; it is a binary operator. Seven theorems on Derived rule are presented, followed by examples of Implicational Theorems. These derived rules of inference start with a hypothesis and a conclusion, if the hypothesis holds then the conclusion holds. Transitive Law of implication and law of commutation involve only implication but no negation, these are presented in a separate sub-section. Logical connectives conjunction, disjunction, and equivalence are outlined. Proofs of theorems with conjunctions, commutativity of conjunction, idempotency of conjunction are explained in a subsection. Symmetry of equivalences, transitivity and associativity of equivalences is followed by proofs and theorems with disjunctions. Commutativity of disjunctions, associativity of disjunctions is proved. DeMorgan's first and second law are stated and proved.

Chapter 2 - Set Theory This is a 62 page chapter. It begins with defining a set. Short examples are provided on binary arithmetic, geometrics, two body problem and three body problem. The three body problem is proved by K F Sundman in 1907. It states that three objects rotating in space will never collide simultaneously. An example each on Geoid and ellipsoid is provided. A survey carried out in the eighteenth century to confirm the shape of the earth is discussed. Its purpose is to demonstrate the difficulty in using "real" objects to illustrate logical and mathematical concepts. Sets do not contain "real" objects instead they contain abstract objects defined by precise rules. This section defines the building block of all mathematics, the elementary stages of set theory is the foundation of all mathematics. Most of mathematics and computer science consists of logical relations between abstract objects called sets. There is no definition of mathematical sets. Sets are foundational objects. All mathematical objects are sets and all quantified variables are sets. Remark 284 states "In designing a proof we may at any stage start from the conclusion - but we may not assume it as True - and then search for logically equivalent formulae that connect the conclusion to other formulae that we know how to prove." The section on mathematical functions is about a domain and range that contain measurements. In the section on Composite and Inverse functions topics such as composition of functions, injective, surjective, bijective and inverse functions are defined. Reflexive, symmetric, transitive relations are discussed. Preorders and partial orders are defined. Zermelo's Theorem and Zorn's Lemma are stated. A lemma is a proposition accepted for immediate use as a proof of another proposition.

Chapter 3 - Induction, Recursion, Arithmetic Cardinality. This is a 63 page chapter. Concepts of numbers, arithmetic counting and "infinity" and their relationship with set theory are shown. Mathematical induction is a method of mathematical proof used for properties of natural numbers. Induction is used in computing and mathematical logic. An extension of mathematical induction is applied in recursion, a programming method that is one of the central ideas in computer science. Recursion relies on the fact that the solution to a problem depends on solution to smaller instances of the same problem. Computation of a factorial is used as an example to demonstrate recursion. A section on Orders and Cancellations explains how natural numbers are used to model geometric and algebraic concepts. The theorem on well ordering principle is proved. Section 3.8 is titled Arithmetic in Finance. The theory explained in this chapter is applied to problems in finance. Topics covered are Percentages and Rates, Sales and Income Tax, Compound Interest, Loans Mortgages and Savings Plans and Perpetuities.

Chapter 4 - Decidability and Completeness Logic and axioms are used for common and scientific patterns of reasoning. The methods used to investigate use of appropriate methods and axioms are discussed. Some logics have the capability to determine if certain formulae are theorems, other logics do not have the capability. Examples in this chapter show all proofs within implicational calculus are not sufficient to prove certain implicational tautologies. Limitations of Truth Tables are shown. The Completeness Theorem provides an algorithm to design a proof for each tautology within full propositional calculus. Well formed sets are discussed. The author differentiates between classical logic and intuitionistic logic. Classical logic is discussed in the first chapter on Logic and Deductive Reasoning. The chapter ends with a section on incompleteness of predicate calculus and Godel's Incompleteness Theorem.

Chapter 5 - Number Theory and Codes This chapter opens with an introduction on codes, its use in messages to provide some level of secrecy and its usage in error detection and correction. A section in Euclidean Algorithm presents the theorem on division with remainder and its proof using induction. The theorem states; for each non-negative integer N and every positive integer M, there exists unique non-negative integers Q and R such that N = (M * Q) + R, M ¿ R. Greatest Common Divisor (GCD) is defined. Two numbers are relatively prime if their greatest common divisor is 1. For example 6 and 25 are relatively prime. The section on GCD covers five theorems. Proof of Euclidean Algorithm is given by use of recursion. The section Digital Expansion and Arithmetic provides a short paragraph on different bases. "Base 60 was used by Babylonian civilization; base 10 was first used in Chinese and Greek civilizations several millennia ago, an combination of base 18 and 20 was used by Mayans several centuries ago." A few examples for conversion from different bases are presented. Section 5.3 covers Prime Numbers. Prime numbers are defined, prime number factorization is explained and an algorithm for primality testing is provided. Only the most fundamental algorithm is presented with two examples. Section 5.4 covers Modular Arithmetic; it includes modular integers, congruence modulo divisors, modulo addition, modular multiplication and division and divisibility tests. Odd and even bases are considered in divisibility tests. Modular code is covered in section 5.5. Examples include International Standard Book Number (ISBN), Universal Product Code (UPC), Bank Identification Code and US 9 digit zip code. Bank Identification Code appears on bank cheques and transaction slips, it consist of a 9 digit decimal number, which includes one check digit. Codes that rely on theorems about remainders of integer division by powers of integers Section 5.6 starts with Fermat's Little Theorem, with proof. An extension of Fermat's Little Theorem, the Euler's Totient function is defined. The section on Euler-Fermat Theorem covers Euler's Totient function and Euler-Fermat Theorem. The last section in this chapter is titled Further Issues in Number Theory. It lists four open problems; twin-prime conjecture, perfect number conjecture, Goldbach's conjecture and Carmichael's conjecture. In Goldbach's terminology 1 is a prime.

Chapter 6 - Ciphers, Combinatorics and Probabilities Cryptography is the science of transforming plaintext into a form that can be read only by authorized users. Plaintext in present times includes images, sound, video, executable programs, ASCII and binary data. Chapter 6 begins by differentiating between codes and ciphers. Many textbooks on cryptography do not make differentiate between the two terms and use them interchangeably. Cryptography relies on the mathematical theory of permutations; it must hide identifiable features of a language such as frequency of letters, diagrams and trigrams. Digrams and trigrams are two letter and three letter words and letter combinations. By give simple examples the author differentiates between permutation and combination. Section 6.1 titled Algebra. A "group" is a mathematical structure consisting of a set with

an operation on any two of its elements to form a third element. Operations include substitutions, addition and multiplication of non-zero rational numbers. Semigroup and monoids are introduced, existence and uniqueness of identities and inverses. Design of successful ciphers codes and other applications depends on correct use of theorems on mathematical groups. Permutation is defined in section 6.2, associativity, inverses and orbits of permutation are discussed.

The simplest of all substitution ciphers, the Caesar's cipher is illustrated with a historical mention of Suetonius. Only 22 letters were used in the Caesar's cipher. Use of cyclic permutations, or cyclic shift, in the ENIGMA is also mentioned. In the section on Factorization of Permutations by Cycles, disjoint cycles and non-disjoint transpositions are discussed. The section on Signatures of Permutations shows how different permutations can correspond to significantly different physical situations, for example, in stereochemistry. Even and odd permutations are illustrated using an example of a lactic acid molecule. The mathematical feature of a signature is explained. Definition of reorders is given. Transposition and signatures as dextrorotatory and levorotatory are discussed using examples of lactic acid. Alternate group of degree n is defined. Section 6.5 is on Arrangements. Arrangements with and without repetition are explained. Examples of pangrams and its use in printing and transmission devices are shown. Pangrams are arrangements with repetitions where each element appears at least once. Panagrams are used to test if printing and transmission devices process each element correctly. The term word is defined as an arrangement with repetition. Section 6.6 is on Combinations. The increase and decrease of Binomial coefficients are presented in a theorem. Examples of palindromes and anagrams are given. Section 6.7 is on Probability. Probability spaces, Exclusion-Inclusion principle, Conditional Probabilities and applications of Binomial Theorem are discussed in this section. Section 6.8 discusses the working of an ENIGMA machine. The alphabet set used, keyboard and display are provided. An example gives further insight in the working. Order of the rotors, ring settings, the plug board, matching of the enciphering and deciphering machines is illustrated. The machine changes the permutation after each letter, this technique prevents repeating patterns in ciphertext for the same pattern in plaintext. A brief history on the ENIGMA and a brief note on the four versions of ENIGMA A, B, C and D are provided.

A subsection in devoted to breaking the enigma ciphers. This section narrates techniques used by code-breakers. Rejewski's first and second theorems and the method to reconstruct the Enigma wiring is presented.

Chapter 7 - Graph Theory This is the last chapter in the book. Applications of graph theory are discussed - such as roads, powerlines, data channels, printed circuit boards and the traveling salesman problem. Diagrams in this chapter are hard to read. They are reduced to a disproportionate size. Figure 7.2 on page 373 is not readable. Graph theory has specialized terms. Author uses terms that have different meaning. A certain class of practical problems is solved by graph theory but larger problems require linear programming. Section 7.1 is on Mathematical graphs, it begins with a subsection on directed graphs. No axioms are presented; its foundation is drawn from set theory and induction. Definitions of directed graphs, vertices nodes, directed graph edge, loop completeness of a directed graph are discussed. Undirected graphs, Reflexivity, Symmetry, transitivity and quotient map are covered. In the section on degree of vertices - inward degree and outward degree are covered. Path Connected Graphs are covered in Section 7.2, topics discussed are Walks in graphs, path connected component of graphs, longer walks and longest paths. A path is a walk that traverses each edge once. In the section on Longest Paths three theorems are presented. Eulerian paths and circuits are covered in a separate subsection. Situations that require origination

and termination at the same point are dealt in this section. A walk is closed if it originates and terminates at the same point. In the section on Mathematical Trees examples are taken from the state of Hawaii. Spanning trees and directed graphs are covered. A section is devoted to Graph Theory in Science. Graphs and multigraphs are discussed with examples of a hydrogen, oxygen, nitrogen and water molecules. Section 7.7.2 illustrates shapes of hydrocarbons.

3 Opinion

The book provides detailed coverage of topics on Logic and Mathematics. The two part presentation of theory and applications is well thought out. The theory is presented in a mathematical style. It could serve as a reference book and a supplementary textbook for an under-graduate course in Logic and Mathematics; the exercises provided are exhaustive. Each chapter ends with a short section titled Project which provides the reader a larger assignment to work on. The book needs a support manual for instructors, with hints, suggestions or complete answers to some problems and exercises; currently the publisher does not have such support material. The diagrams in the book need to be redone; some of them are not readable. The index needs to be more complete by including all symbols and notations used in the book. The treatment of examples is terse, students would benefit from a detailed presentation. It was a pleasure reviewing the book.

Review of[3] of
Rippling: Meta-Level Guidance For Mathematical Reasoning
Cambridge Tracks In Theoretical Computer Science 56
by Alan Bundy, David Basin, Dieter Hutter, and Andrew Ireland
Cambridge University Press, 2005
202 pages, HARDCOVER

Review by
Dr. Maulik A. Dave, maulikadave@aim.com
4333 Dunwoody Park, Apt. 3107, Dunwoody, GA 30338.

1 Introduction

Theorem proving by machine is a very well developed area. The book is about Rippling, which is a proof planning technique. Automatic theorem proving is done by rewrite rules. The assumptions and already proved theorems are supplied to an automatic theorem prover in the form of rewrite rules. The prover program attempts to prove the conjecture by applying the rewrite rules given. It may enter into a combinatorial explosion search, which may be impractically time consuming. Rippling provides meta level guidance to the prover based on proof planning. In proof planning, plans are made to try out various methods to be tried. If one plan fails, the failure information is analyzed to come up with a better plan. When the rippling method is used for proof planning, the software implementing rippling is given to the guidance plan. The guidance is provided by annotating the rewrite rules, and wave rules. The wave rules are rules which have been annotated. The rippling program comes back quickly either successfully proving or failing. In other words, rippling takes givens and goals. Givens are formulas, which are assumptions, or previously proved theorems. The goals are the conjectured formulas to be proved from givens. Rippling finds out the differences between the givens and goals, and moves out the differences so that goals contain instances of givens. When goals contain instances of givens, the conjectures may be proved; however, further work may be required. If rippling is not able to do this, then the results where rippling failed are analyzed. This analysis is used to find a way to repair the failed proof. Most common example, where rippling can be used, is inductive proofs. Rippling still terminates even when rewrite rules are used both right-to-left and left-to-right.

The following example is from the book, which can be proved by rippling:

Given: $a + b = 42$

Goal: $((c + d) + a) + b = (c + d) + 42$

Another example given in the book is about the linked list. Rippling proves the distributive law of reverse over append by induction. The base case is trivial. The step case rippling solves:

Let rev is a list reversing function. Let $<>$ be the infix list appending function. Let :: be the binary infix function.

Given: $rev(t <> l) = rev(l) <> rev(t)$

Goal: $rev(h :: t <> l) = rev(l) <> rev(h :: t)$

[3]

An example of addition of even numbers is also proved by rippling. Let $even(i)$ be a function that returns YES if i is an even natural number, and NO otherwise. Rippling proves that $even(m)$ and $even(n)$ implies $even(m + n)$.

2 Summary

The book has 7 chapters. Apart from a chapter on introduction, and a chapter on conclusions, it has chapters on the varieties, theory, and scope of rippling. It also has a chapter on failure analysis, and a chapter on general methodology.

The first chapter on introduction begins by introducing rippling, and proof planning in the context of the problems of automating reasoning, and formal methods of system development. It introduces rewrite rules. It informally explains rippling by examples expressed as actual rewrite rules, and annotations on rewrite rules. Patterns formed by application of wave rules are referred to as wave fronts. The chapter ends with a short section on the history of rippling.

The second chapter is on varieties of rippling. It begins with compound wave fronts, which are wave fronts with more than one functional applications. For simplification, the wave rules are kept in maximally split normal form. The chapter then discusses movements in the rippling with respect to givens, and goals such as sideways rippling, inward ripplings, and weak fertilizations. It also discusses conditional wave rules. The chapter ends by introducing concepts of higher order rippling, rippling to prove equations, and relational rippling.

The third chapter is on how to use the information when rippling fails. It discusses revision of inductions, lemma discovery, lemma alternatives, and conjecture generalization by analyzing rippling failures. The chapter ends by showing tables generated by a practical proof planner, which has implemented these techniques.

The rippling formalism is presented in the fourth chapter. The chapter begins by mathematical preliminaries of term rewrite. The rippling is formalized as a binary relation over the set of the annotated terms. Erasure, and skeleton functions are defined over annotated and unannotated terms. The formalism puts 3 requirements: If a term ripples to another term, then erase of the term rewrites to erase of another term; the skeletons of another term are subsets of skeletons of the term; Rippling terminates. It then discusses various methods to achieve the requirements, and the associated proofs. The chapter ends with a discussion on dynamic wave rule parser method.

The fifth chapter begins by presenting practical examples, where rippling is successful. The examples include bi-conditional decision procedure, summing of the binomial series, and lim + theorem. It is mentioned that rippling is successfully applied in reasoning about imperative programs, and framework logics. The chapter then presents some examples such as mutual recursion, inverting a tower, and difference removal, where rippling has failed. It ends with remarks that CLAM, and INKA, where rippling is implemented, are used in various applications.

The sixth chapter is on methodology. It begins by describing a general annotation language. It presents some examples of encoding constraints on proof search. The examples explain various strategies of encoding constraints. The strategies are rippling, paramodulation, window inference, and reuse. Then, the chapter discusses abstraction approach, where representations of the problems are simplified by abstraction. The strategies are also discussed with respect to this approach.

The seventh chapter summarizes the book. There are 2 appendixes. The first appendix shows an annotation calculus and a unification algorithm for it. The second appendix presents functions given in the book.

3 Opinion

The students of theoretical computer science will find the book useful. A background of theorem proving and term rewrite systems is required to understand the book. To understand the examples, knowledge of various data structures, algorithms, and mathematical theorems is required. A course on theorem proving, or proof planning techniques will have this book listed as one of the textbooks, or references. For users of the systems, where rippling is implemented, the book gives a good number of illustrations. The book shows that the Edinburgh CLAM Family, a proof planning software system implements the rippling. It also shows that Saarbruken INKA software system for theorem proving, implements rippling. VSE, a tool for formal software development, integrates INKA. Examples of applications developed using VSE include a scheduling system for a distributed radio broadcasting, security models for various smartcard applications, etc. Many proof obligations of such systems were completed using rippling as mentioned in the book. Loop invariant verifications were also completed using rippling in some applications.

The book will be also useful for building a proof planning tool.

Review of[4]
Handbook of Nature-Inspired and Innovative Computing
Edited Albert Y. Zomaya
Springer, 2006
736 pages, hardcover

Review by
Aaron Sterling `sterling@iastate.edu`
Department of Computer Science, Iowa State University[5]

1 Introduction

The ambitious goal of the *Handbook of Nature-Inspired and Innovative Computing* is to "to be a Virtual Get Together of several researchers that one could invite to attend a conference on 'futurism' dealing with the theme of Computing in the 21st Century." The Handbook contains 22 chapters, written in a "workshop style," meaning that little to no background is required from the reader except for basic knowledge of computer science, and the chapter authors provide an overview of a particular research area. The material is divided into three sections: Models (i.e., theory), Enabling Technologies (i.e., hardware), and Application Domains (i.e., recent, novel applications of computer science). Most of the chapters present either theoretical models (fuzzy logic, quantum computing, swarm intelligence) or report on trends in non-von-Neumann-model hardware (morphware, optical VLSI, neural models in silicon). Overall, I found this volume to be a fascinating, and "gentle," introduction to a wide range of nonstandard research areas of computer science.

2 Summary

2.1 General overview

Rather than attempt to discuss all 22 chapters, I will make some general remarks about the material in each of the three sections, and consider a few chapters in detail to provide a flavor for the book's contents. To begin, though, here are three quick points.

First, there are "mandatory" chapters on Bioinformatics (by Aluru) and Quantum Computing (by Eisert and Wolf). These are well written, but there is extensive tutorial-level material on both of these subjects available many places on the internet. So the strength of the Handbook, in my eyes, lies more in the introduction it provides the reader to less-emphasized research directions. Second, as a theorist, I found the chapters on *hardware* to be the most exciting, because they were accessible to me, even with very limited background, and they discussed areas I had never encountered before. So I would encourage SIGACT News readers to consider that a potential strength of the Handbook. Third, the Handbook was published in 2006, so some of the material it contains is dated, but not in an "obvious" way. For example, there is a chapter on fuzzy logic (by Taheri and Zomaya) which is fine, but the 2008 paper by Zadeh titled, "Is there a Need for Fuzzy Logic?" [Information Sciences

[5]This review was written while the author was visiting the Department of Electrical Engineering and Computer Science, Northwestern University. This work was supported in part by NSF grant CCF-1049899.

178 (13), pp. 2751–2779, 2008] is much more comprehensive, in my eyes. On the other hand, the chapter "Trends in High-Performance Computing," (by Dongarra) is still completely relevant, except for the most recent numbers themselves on current speeds of supercomputers: the trends described still hold. So, a theory chapter is "more dated" than a hardware chapter. My suggestion, therefore, is for the reader to consider the chapters as a jumping-off point to learn about unfamiliar areas, rather than considering them to be the "definitive word" on any subject.

2.2 Section on Models

The Models section of the Handbook contains several chapters whose computational paradigms could be loosely categorized as "lots of small agents act locally to solve large, global problems." These paradigms include collaborative algorithmics (by Rosenberg), a hybrid association algorithm (by Tari and Wu), multi-set rule based soft computing (by Krishnamurthy and Krishnamurthy), evolutionary computing including genetic algorithms (by Seredynski), and swarm intelligence (by Kennedy). For reasons of space, I will consider one of these: the chapter on swarm intelligence.

James Kennedy, author of the chapter "Swarm Intelligence," focuses on *particle swarm optimization*. Informally, a particle swarm is a population of agents who communicate according to some topology of connections, and a change rule. The "classic" particle swarm topologies have been everyone-connected-to-everyone, and a ring lattice (where each agent has two neighbors). There are many different change rules in the literature, but most are based on a standard formula, and the basic idea is that a particle chooses where to go based on its success (or failure) at the previous time step, and the success (or failure) of its neighbors. Unlike searches performed by evolutionary algorithms, which first find "good" regions, and then try to ramify the good regions into best possible points, a particle swarm continues exploring the entire space until the algorithm concludes. As a result, in some ways, particle swarm algorithms are more flexible than evolutionary algorithms, because they assume less about the structure of the search space. The author discusses technical differences between particle swarm algorithms, and when to prefer one over the other. He also draws an intriguing line between evolutionary algorithms (such as genetic algorithms) and social algorithms (such as particle swarm or ant colony optimization): to optimize an n-dimensional problem with a "traditional" evolutionary algorithm requires a $2n$-dimensional vector (each function parameter, plus a standard of deviation variable for each parameter), whereas a social algorithm requires only an n-dimensional vector, because the "mutation" in the social algorithm is determined by the distance between the individual and the source of its social influence. The author concludes with a section on non-computer applications of particle swarms, including a pharmaceutical company that used particle swarm optimization in petri dishes to find an optimal mixture of ingredients for growing bacteria.

2.3 Section on Enabling Technologies

The Enabling Technologies section of the Handbook begins with a chapter that introduces "classical" computer architecture (written by Yi and Lilja), then a chapter on the history of optical VLSI (by Eshagian-Wilner and Hai), and then moves into less traditional areas. There are chapters on morphware (by Hartenstein) and the related topic of evolving hardware (by Gordon and Bentley). There is a chapter on implementing neural models in silicon (by Smith), and molecular and nanoscale computing (by Eshagian-Wilner et al.). The chapter by Dongarra on high-performance computing that I mentioned above appears in this section, as do three chapters whose topics would

probably be grouped under "cloud computing" today: cluster computing (by Yeo et al.), web service computing (by Benatallah et al.), and predicting grid resource performance (by Wolski et al.). I will now summarize Hartenstein's chapter "Morphware and Configware."

Reiner Hartenstein's chapter on morphware and configware is one of the most extensive in the Handbook, and reads almost like a mini-encyclopedia, with 27 figures, including a glossary of over 50 hardware acronyms. The author begins by arguing that the traditional hardware model, the so called von Neumann paradigm, where microprocessors driven by software instructions communicate with accelerators driven by data, is facing a design crisis: as a System on a Chip becomes more sophisticated, the silicon wafers required for its design are becoming prohibitively expensive. The solution: morphware. "Morphware is structurally programmable hardware, where the interconnect between logic blocks and/or functional blocks, as well as the active functions of those blocks, can be altered individually by downloading configware, down to the configuration memory (configuration RAM) of a morphware chip." (The most ubiquitous example of morphware is probably the field-programmable gate array (FPGA), which is a multi-billion-dollar industry.) The author discusses different morphware paradigms and their recent impact on the computing sciences. He concludes with a call to reform computer science curricula with a new taxonomy of architecture and algorithms. His concern is that undergraduates who develop a procedure-only mindset will be at a tremendous disadvantage on the job market—and I agree.

2.4 Section on Application Domains

The final section of the Handbook, on Application Domains, is the shortest. It contains the Bioinformatics chapter (by Aluru) I previously mentioned, and chapters on pervasive computing (by Kumar and Das), information display (by Eades et al.), and noise in foreign exchange markets (by Szpiro). I will summarize the chapter on pervasive computing.

Mohan Kumar and Sajal K. Das contributed the chapter "Pervasive Computing: Enabling Technologies and Challenges." In their view, "pervasive computing is about providing 'what you want, where you want it, when you want it, and how you want it' services to users, applications, and devices." Technologies in this category include sensors, RFID tags, wearable computers and smart phones. The authors spend five pages providing a snapshot (circa 2005) of the state of these technologies—for example, how objects are tracked using MEMS (micro-electro-mechanical systems), or RFID (radio frequency identification). They then spend five pages discussing the following challenges of pervasive computing: (1) heterogeneity and interoperability, (2) proactivity and transparency, and (3) location awareness and mobility. Challenge (1) could be phrased, "How can so many different kinds of devices seamlessly communicate without need for human interaction?" The point of challenge (2) is for computing technologies to be increasingly aware of situational needs of users, so they immediately provide the right category of data. The authors give an example of a doctor in a cafeteria who receives a message on his PDA. The cafeteria server needs the doctors profile, but also needs to receive the doctor's schedule from the PDA, so the server will know whether it is an appropriate time to notify the doctor about the message. The authors break challenge (3) into several subchallenges, such as prediction of future location, scalable signaling traffic, and security issues. The chapter concludes with sketches of four pervasive computing projects, including the MavHome Smart Home, whose objective is to decompose control of an entire house into four computing layers.

3 Opinion

As I stated in the introduction, the chapters of the *Handbook of Nature-Inspired and Innovative Computing* that I got the most from were the chapters on architecture, not theory. I already had (limited) background in, e.g., genetic algorithms, neural networks, fuzzy logic. On the other hand, the practical and theoretical challenges posed by the current market for evolutionary hardware and pervasive computing were brand new to me – and quite exciting! So I believe theorists looking for interdisciplinary applications for TCS might benefit from reading this book. (Similarly, applied computer scientists might benefit from a broad introduction to newer areas of CS theory.)

I could see this book used as the basis for a seminar on approaches to decentralized computation. It contains no exercises, so it isn't appropriate as a textbook. Most of the chapters could be read by a clever advanced undergraduate, so I consider it a collection of tutorials, rather than of research monographs. Overall, I think this book would be of use to anyone with some CS sophistication who desires exposure to a wide range of newer or less-publicized research areas.

Review of[6]
Algorithms and Data Structures: The Basic Toolbox
by Kurt Mehlhorn and Peter Sanders
Springer, 2008
300 pages, Hardcover

Review by
Hal C. Elrod (helrod@acm.org)

1 Introduction

A toolbox: in the hands of an accomplished craftsman, it has a carefully selected set of the most commonly-used implements appropriate to the most frequently-encountered tasks. In the hands of a hack, it is a box of battered junk misapplied to whatever nail, screw, or pipe in sight.

What are the basic tools of the accomplished programmer? In the classic title by Nicolas Wirth, Algorithms + Data Structures = Programs[1]. Along with a compiler and `vi`, the tools in this book should cover most of the interesting problems we face.

But in the real world of commercial software development, few of us spend time writing optimal stacks and queues, or tweaking the runtime of sorting algorithms. Widely used languages such as C++ and Java support such structures via built-in or standard libraries—the Java library and C++ Standard Template Library (STL), respectively. If not included with the language, the prevalence of open-source implementation libraries such as Boost[2] and JDSL[3] have made the foundational building blocks widely accessible. What is needed, and what *Algorithms and Data Structures* attempts to provide, is a understanding behind the libraries. An understanding not just into implementation details but into the mathematical underpinnings that make these things work. If successful, it will be just the right tool when that library *qsort* exhibits $O(n^2)$ behavior.

The authors are well-known in the study and practice of computer science. Kurt Mehlhorn is an ACM Fellow and one of the developers of LEDA, the Library of Efficient Data types and Algorithms[4]. Co-author Peter Sanders has contributed heavily to the field of randomized algorithms, among other topics, and is one of the original authors of `STXXL`, the Standard Template Library for Large Data Sets[5]. Both LEDA and `STXXL` are examples of libraries that efficiently implement some of the algorithms and data structures referenced in the book, but none of the examples or pseudo-code require any particular library. Rather, each chapter concludes with Implementation Notes discussing common library implementations in C, C++, and Java, which helps balance the book towards the practical rather than purely theoretical.

2 Summary

The book has 12 chapters, along with an Appendix listing mathematical symbols, concepts, and useful formulae.

Chapter 1: Integer Arithmetics Cleverly, Chapter 1 precedes the introduction, and sets the table with basic math. It starts with schoolbook addition and works through the Karatsuba method of integer multiplication, including a simple C++ implementation.

Chapter 2: Introduction The introduction proper starts with a discussion of asymptotic ("Big O") notation, the RAM model, and the elements of pseudo-code. Then these basics are applied to an increasingly complex set of algorithms and data structures: linear search, binary search, randomized algorithms, graphs and trees. Finally, **P** and **NP** are outlined briefly.

Chapter 3: Representing Sequences by Arrays and Linked Lists This data structure chapter starts with the ever-popular doubly and singly-linked lists, then on to arrays, stacks, and queues. "Ever-popular" in that every undergraduate must learn them, and every professional writes one once. He or she then either reuses that version or, if a truly excellent professional, realizes that there are better generic implementations in the STL or Java *util* package. More important is to understand *why*, and to that end the book introduces the concept of Amortized Analysis. Every data structure has trade offs; a doubly-linked list is fast to concatenate, but slow to address a specific item, an array is fast to address, but slow to concatenate. Answering the *why* with a degree of mathematical rigor is what makes this book a quality tool. An example of the practical nature are the references for each algorithm to the C++ STL and Boost libraries and Java library implementations.

Chapter 4: Hash Tables and Associative Arrays Covers hashing, collisions, hashing with chaining, hashing with linear probing—the basics of hashing, followed by an explanation of universal hash functions. In a clever, non-technical example of a hashing the authors describe the following:

> If you want to get a book from the central library of the University of Karlsruhe, you have to order the book in advance. The library personnel fetch the book from the stacks and deliver it to a room with 100 shelves. You find your book on a shelf numbered with the *last* two digits of your library card. Why the last digits and not the leading digits? Probably because this distributes the books more evenly among the shelves. The library cards are numbered consecutively as students sign up, and the University of Karlsruhe was founded in 1825. Therefore, the students enrolled at the same time are likely to have the same leading digits in their card number, and only a few shelves would be in use if the leading digits were used.

Starting in this chapter, the exercises get a little more difficult. The book contains exercises for each major section, ranging from the trivial to the mathematically challenging. Answers are not provided in the text.

Chapter 5: Sorting and Selection Selection, mergesort, and quicksort are described algorithmically and analyzed as to expected and worst-case performance.

Chapter 6: Priority Queues Introduces heaps in their binary, Fibonacci and other variations.

Chapter 7: Sorted Sequences Covers binary search trees, *(a,b)*-trees and Red-Black trees.

Chapter 8: Graph Representation This chapter sets the stage for the algorithmic chapters that follow by covering the most common graph representations: arrays (static graphs), adjacency lists and matrices.

Chapter 9: Graph Traversal Breadth-First Search (BFS), Depth-First Search (DFS).

Chapter 10: Shortest Path Describes and analyzes Dijkstra's simple yet sublime shortest path algorithm, and other algorithms for graphs with arbitrary (negative) edge costs.

Chapter 11: Minimum Spanning Trees Defines and then explores MST with Jarník-Prim and Kruskal's algorithms.

Chapter 12: Generic Approaches to Optimization This chapters covers the major categorizes of optimization problems, starting with the knapsack problem through linear and integer programming, then on to greedy heuristic approaches, then quickly through simulated annealing, Tabu Lists, and other attacks on challenging problems. In keeping with the practical and compact nature of the book, these approaches are covered descriptively to introduce the types of approaches, but not in every detail. To quote one example:

> In fact, although LP solvers are used on a routine basis, very few people in the world know exactly how to implement a highly efficient LP solver.

And in fact, the book directs the reader to view LP as a "black box", which makes sense in the real world, where highly optimized solvers are available for free download. Unless one is a specialist in the field, it is pointless to write an LP solver from scratch, and I speak from having written one.

3 Opinion

A "Toolbox" should be portable, practical, and useful. This book is all these, covering a nice swath of the classic CS algorithms but addressing them in a way that is accessible to the student and practitioner. Furthermore, it manages to incorporate interesting examples as well as subtle examples of wit compressed into its 300 pages. Although it is not tied to any one language or library, it provides practical references to efficient open-source implementations of many of the algorithms and data structures; these should be the first refuge of the commercial developer. I can easily recommend this book as an intermediate undergraduate text, a refresher for those of us who only dimly remember our intermediate undergraduate courses, and as a reference for the professional development craftsman.

Notes

[1] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice Hall, 1976

[2] www.boost.org

[3] www.jdsl.org

[4] www.algorithmic-solutions.com/leda LEDA is available as object code in Free Edition, but source is licensed separately.

[5] stxxl.sourceforge.net `STXXL` is free/open source.

Review of
The Algorithm Design Manual, Second Edition[7]
Author: Steven S. Skiena
Publisher: Springer-Verlag, 2008
ISBN NUMBER: 978-1-84800-069-8
730 pages, Hardcover, $61.00

Reviewer: Neelakantan Kartha (gnkartha@yahoo.com)

1 Overview

The Algorithm Design Manual by Steven Skiena is aimed at two groups of people: students and professionals. It consists of two parts: (I) Practical Algorithm Design and (II) The Hitch Hiker's Guide to Algorithms. The first part gives an overview of standard algorithm analysis and design techniques, including worst case analysis, data structures, graph algorithms and dynamic programming. This part will also be of interest to the practitioner who wants to quickly brush up his knowledge in a particular area. The second part is a annotated catalog of algorithms that allows one to quickly zoom in an algorithm of interest. The description of each algorithm includes a discussion section that addresses common questions and points to implementations and related problems.

2 Summary of Contents

Part I of the book focuses on algorithm design and consists of ten chapters. It starts off with an introductory chapter introducing algorithms and motivating why the analysis of algorithms is important. The introductory examples on robot tour optimization and jobs scheduling serve to nicely illustrate why algorithm correctness is a property that must be demonstrated.

The second chapter on algorithm analysis is more or less standard: it defines and illustrates best, worst and average case complexity and gives several examples of working with the Big Oh notation.

The third chapter on data structures introduces several standard data structures, including lists, stacks, queues and dictionaries. This chapter should mostly be a review for someone who has taken a previous course in data structures.

The fourth chapter focuses on sorting and searching, and introduces heap sort, merge sort, quick sort and binary search. Analysis of these algorithms is presented in an informal manner.

The fifth chapter on graph traversal discusses data structures on graphs and breadth and depth first search. It also discusses some applications of breadth and depth first search, such as finding cycles and strongly connected components.

The sixth chapter on weighted graph algorithms presents Prim's and Kruskal's algorithms for minimum spanning trees and Dijkstra and Floyd's algorithms on shortest paths. It also includes a brief introduction to network flow algorithms.

The seventh chapter on combinatorial search and heuristic methods introduces backtracking and applies it to several problems. It also contains a discussion of local search and simulated annealing.

The eighth chapter introduces the important dynamic programming technique of caching intermediate values of computation for speedup in run time and illustrates this idea on a number of examples, such as the computation of the edit distance between two strings.

The ninth chapter on intractable problems introduces the theory of NP-completeness and gives techniques on proving hardness using reductions.

The tenth chapter gives a list of questions that helps guide the design of an algorithm for a problem of interest.

Part II of the book is an annotated catalog for algorithms for solving different types of problems. It contains a number of algorithms for solving numerical, combinatorial, graph, computational geometry and set and string problems.

3 Opinion

The Algorithm Design Manual is different from the typical algorithm texts such as *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein. First of all, it focuses more on the practical aspects of algorithm design, without laying much stress on the underlying mathematical analysis. It is written in an informal style that I found pleasant and engaging. Considerable effort has gone into making the algorithm guide in Part II useful, by giving a pictorial description of the problem being solved, by presenting variants of the basic algorithm when appropriate and by giving references to implementations. The book's unique structure makes it more likely to be immediately useful to the practitioner who has problem to solve and wants to quickly make progress, without getting bogged down in unwanted detail or mathematical theory. In that respect, the book succeeds admirably.

However, I do have some reservations in using the book as a primary text for algorithm classes. Here are a few nits: It would have been better to present the insertion sort in chapter 1 in pseudocode (similar to what is done in Section 1.1) so that the beginning student is not distracted by the syntax of the C language right at the beginning. The war stories are interesting and illuminating, but some of them (such as the ones in Section 1.6 and Section 3.6) require more detail to be completely clear. A more detailed presentation of asymptotic analysis would be helpful to the student who has never seen this material before. Certain code fragments are not self-contained (for example, `bubble_down` function on pg. 114 uses `pq_young_child` function, which is not defined). And to make the book suitable for self study, presenting answers to a subset of the exercises, and pointing out particularly instructive ones would help. I also noticed a few typos (e.g., displayed formula at the bottom of pg. 17 (the lower index should be 1), pg. 85 (line 15), pg. 109 (line 15: the reference should be to page 42)).

Overall, I recommend this book warmly.

Review of [8]
Graph Theory: A Problem Oriented Approach
Daniel Marcus
MAA, 2008
205 pages, Hardcover

Review by
Haris Aziz (aziz@in.tum.de)
Technische Universität München, 85748 Garching, Germany

1 Introduction

Graph Theory: A Problem Oriented Approach is a textbook cum workbook with the goal to assist a proactive and problem-oriented approach to learning graph theory. This goal is clear from the offset as many key concepts, examples and even proof arguments are clarified or highlighted by the help of carefully chosen problem questions. A reader unfamiliar with graph theory is not kept in the dark as problems are preceded by some introductory text which nudges the reader along. The book consists of over three hundred strategically placed problems interspersed by necessary text.

There are also further problems to think about at the end of each chapter. The book is written in a similar style as a book on combinatorics by the same author [2] .

2 Opinion

The first thought that springs to mind is that the book is fairly complete in terms of coverage of topics. The main topics covered include spanning tree algorithms; Euler paths; Hamilton paths and cycles; planar graphs; independence and covering; connections and obstructions; vertex and edge colorings; matching theory; cycle-free digraphs; and network flow theory.

Another aspect of the book is that students are familiarized with algorithmic issues. Key ideas of standard graph algorithms are highlighted. Algorithms covered include *Prim* and *Dijkstra's* algorithms to compute the minimum spanning tree; the *Hungarian algorithm* for the *optimal assignment problem*; and the maximum flow algorithm. The algorithmic treatment is not as advanced as for example in [1]. The book is supplemented by a useful index which lists the name of key terms, theorems and algorithms.

The focus of the book is on self-discovery. The author compromises formality of notation and definitions and focuses on highlighting key graph theoretic concepts in a natural and thought-provoking way. Apart from problem questions, the text comprises of informal definitions and exposition of concepts.

3 Conclusion

One expects the book to be a textbook cum workbook but it veers towards being a very carefully planned workbook with minimal text. The book can be used to plan and induce interaction in a

first year undergraduate graph theory class. Even high school students can pick up some valuable concepts by independent self-study. Although, an excellent teaching and learning resource, the book is not detailed enough to constitute an independent textbook. Therefore, it is more suited to complement a standard graph theory textbook for an undergraduate class.

References

[1] A. Gibbons. *Algorithmic graph theory.* Cambridge University Press, Cambridge ; New York, 1985.

[2] D. A. Marcus. *Combinatorics: a Problem Oriented Approach.* Cambridge University Press, Cambridge ; New York, 1998.

Review of[9]
Proofs from THE BOOK (4th edition)
by Martin Aigner and Günter M. Ziegler
Published by Springer 2009
274 pages, Hardcover
Amazon: $40.00 new, $50.00 used[10]

Review by Miklós Bóna

1 Introduction

Paul Erdős, the most prolific mathematician of the twentieth century, had his own way of judging the beauty of various proofs. He said that there was a book somewhere, possibly in heaven, and that book contained the nicest and most elucidating proof of every theorem in mathematics. When somebody showed him a new proof, he either said, "this is the Book proof" or "this is not the Book proof".

Martin Aigner and Günter Ziegler succeeded admirably in putting together a broad collection of theorems and their proofs that would undoubtedly be in the Book of Erdős. The theorems are so fundamental, their proofs so elegant, and the remaining open questions so intriguing that every mathematician, regardless of specialty, can benefit from reading this book. The book has five parts of roughly equal length.

2 Number Theory

The first section, on Number Theory, starts with six proofs for the existence of infinitely many primes. At least two of them will be new for most readers. The last one proves the considerably stronger fact that $\sum_{p\in\text{primes}} \frac{1}{p}$ is divergent. Next, the authors explain Bertrand's postulate that states that if $n > 1$, then there is a prime number between n and $2n$. They also mention the much stronger statement (still unproven) that there is a prime between n^2 and $(n+1)^2$ as well.

Leaving prime numbers, we next prove that binomial coefficients are almost never powers, a statement that is very important in coding theory. The precise theorem is that the equation

$$\binom{n}{k} = m^l$$

has no integer solutions for $4 \leq k \leq n-4$ and $l \geq 2$.

After this, we spend the next section deciding which positive integers can be written as the sum of two perfect squares. The answer is that exactly those whose prime factors of the form $4k+3$ all appear with even exponents in the prime factorization of n. For instance, $153 = 3^2 \cdot 17$ satisfies this requirement, and indeed, it is the sum of $153 = 12^2 + 3^2$.

At first sight, the next topic seems to be an outlier since the statement itself belongs to abstract algebra, rather than number theory. A *division ring* is a ring with a unit element in which every nonzero element has a multiplicative inverse. So division rings would be fields if multiplication

[9] ©2011, Miklós Bóna

[10] This is not a typo- it is listed as costing more used then new. That is a paradox with no book resolution.

were commutative in them. The classic theorem presented here is that *finite* division rings actually *are* fields. The proof, while not easy, uses only elementary number theory, so the placement of the theorem in this part of the book is appropriate after all.

The number theory part of the book ends by a section on irrationality results. The two most classic results of that kind are that both e and π are irrational. Besides these, we are shown proofs of the stronger results that π^2 is irrational, and e^r is irrational for all nonzero rational numbers r. An elegant proof of the well-known identity

$$\sum_{n\geq 1} \frac{1}{n^2} = \frac{\pi}{6}$$

is presented, together with a proof of the less well-known fact that

$$\frac{1}{\pi} \arccos\left(\frac{1}{\sqrt{n}}\right)$$

is irrational for all odd integers $n > 1$. Now comes the only critical remark in this review. This reviewer would have liked to see some of the extremely difficult open questions related to e and π here, such as whether their sum or product is transcendent or algebraic. Some of these questions have been open for decades.

3 Geometry

The second part of the book is about Geometry. In two dimensions, the Bolyai-Gerwien theorem shows that if two polygons have the same area, then they are both *equidecomposable* (can be dissected into congruent triangles) and *equicomplementable* (can be made congruent by adding congruent triangles. Hilbert's famous third problem was asking for tetrahedral counterexamples showing that the same is not true in three dimensions. The authors present the Dehn-Hadvigger theorem, which provides sufficient condition for two polyhedra to be *not* equicomplementable, hence not equidecomposable.

Next are some theorems from combinatorial geometry. First, it is shown that for any set S of $n \geq 2$ points, no three of which are collinear, there exists a straight line that contains exactly two points of S. (Hint: consider the pair (P, e) so that $P \in S$, and e is a straight line connecting two points of S, and the distance of P and e is minimal. Then e contains exactly two points of S.) Just as interesting is the *slope problem*, claiming that n points on the plane, no three of which are collinear, always determine at least $n-1$ different slopes. Equality is only possible for odd numbers n larger than four.

Then we get to Euler's famous theorem on connected planar graphs. If such a graph has n vertices, e edges, and f faces, then

$$n + f = e + 2.$$

This theorem has a plethora of consequences. If G is a simple planar graph, then it has a vertex of degree at most five. The inequality $e \leq 3v - 6$ holds for G. If the edges of G are two-colored, then there is a vertex of G with at most two color changes in the cyclic order of the edges around the vertex. The latter leads to the following interesting fact. Given any collection of white and black points on the plane, there is always a straight line that contains at least two points of one color, and none of the other.

Another consequence of the edge-coloring result of the last paragraph is *Cauchy's Rigidity theorem.* This theorem states that if the *convex* polyhedra P and P' are combinatorially equivalent, (meaning that there is a bijection between their face sets that preserves edge containment) and the corresponding faces are congruent, then P and P' themselves are congruent. This theorem does not hold if the convexity condition is dropped.

Then comes an excursion to higher dimensional geometry. Let $f(d)$ be the highest number so that $f(d)$ simplices of dimension d can be placed in R^d so that their pairwise intersections are $(d-1)$-dimensional. It is conjectured that $f(d) = 2^d$. The authors show that $2^d \leq f(d) \leq 2^{d+1}$. Staying in higher dimensions, they then show that in any $2^d + 1$-element set of points in R^d, there are three points that span an angle larger than $\pi/2$ (an obtuse angle).

The part on Geometry ends by Borsuk's conjecture and its disproof. This conjecture stated that every set $S \subset R^d$ of bounded diameter could be partitioned into at most $d+1$ sets of smaller diameter. We are shown a counterexample in dimension $d = 861$ Interestingly, it involves prime numbers. It is known that for the claim of the conjecture to hold, we must partition S into more than $(1.2)^{\sqrt{d}}$ sets of smaller diameter in the general case, but 1.25^d smaller sets suffice.

4 Analysis

The third part of the book discusses problems in Analysis. The first of its sections is on set theory. It introduces the notions of countable and uncountable sets, and proves that the set of all real numbers is uncountable, meaning that there is no bijection from the set of integers to the set of reals. It also explains the continuum hypothesis. Set theorists were wondering if there existed a set B that is strictly larger than the set of natural numbers, but strictly smaller than the set of real numbers. The authors explain, that the answer to this question depends on what system of axioms we use for our proofs involving real numbers: for certain system of axioms, such a set B will exist, for certain others, no such B will exist.

Next, we are shown a collection of inequalities that even a good high-school student can appreciate. The first on the list is the Cauchy-Schwarz inequality, then the inequalities between harmonic, geometric, and arithmetic means. Finally, we are shown a higher-level application, originally proved by Laguerre. It shows that if all roots of a polynomial $x^n + a_{n-1}x^{n-1} + \cdots + a_0$ are real, then the roots are contained in the interval whose endpoints are

$$-\frac{a_{n-1}}{n} \pm \frac{n-1}{n}\sqrt{a_{n-1}^2 - \frac{2n}{n-1}a_{n-2}}.$$

This is followed by a very simple proof for the Fundamental Theorem of Algebra, that is, the fact that any non-constant polynomial with complex coefficients has at least one complex root. We are also shown Monsky's theorem that claims that it is impossible to cut up a square into an odd number of triangles of equal area. It is remarkable that this theorem, which seems to be a theorem from Combinatorial Geometry, has an analytic proof.

Then come sections about less widely known results. The first is the following theorem of Pólya. Let $f(z)$ be a polynomial with complex coefficients, degree at least 1, and leading coefficient 1. Let $T = \{z \in C : |f(z)| \leq 2\}$. Take any line L in the complex plane, and project T orthogonally to L. Then the total length of this projection is never more than 4. The second one is a result of Littlewood and Oxford, which is solved using Sperner's theorem, a result in Combinatorics which

has its own turn in the book, in a later part. The result that is proved here is that if $a_1, a_2, \cdots, a_n$ are complex numbers, $|a_i| \geq 1$ for all i, then the number of linear combinations

$$\sum_{i=1}^{n} a_i \epsilon_i$$

with $\epsilon_i = \pm 1$ that lie in the interior of any circle of radius 1 is at most $c2^n \frac{\log n}{\sqrt{n}}$. A key observation in the proof is that $\binom{n}{n/2} \leq \frac{\log n}{\sqrt{n}}$.

The chapter closes with two classic results, one is the stunning formula

$$\pi \cot \pi x = \lim_{N \to \infty} \sum_{-N}^{N} \frac{1}{x+n}$$

whose proof is called the *Herglotz trick*. The other one is the problem that led to the theory of Monte Carlo approximation methods. If we drop a needle of length l on a paper that is ruled with equally spaced lines of distance $d \geq l$, then the probability that the needle will rest in a position when it crosses one of the lines is $\frac{2l}{\pi d}$. This formula has been used to approximate π for centuries. We are shown a proof with, and one without, integrals.

5 Combinatorics and Graph Theory

The last two parts, Combinatorics and Graph Theory, contain the most spectacular proofs in the book. That sentence should not come as a surprise, given that both authors are combinatorialists (and so is this reviewer).

The Combinatorics part starts with the fundamental ideas of the Pigeon-hole Principle, and Double counting, that is, the idea that if we count certain objects in two different ways, the result should be the same. We are shown simple proofs for the facts that if we choose $n+1$ integers of the set $\{1, 2, \cdots, 2n\}$, then there will be two chosen integers that are relatively prime to each other, but there will also be two so that one divides the other. A higher-level application is Sperner's lemma, which is a statement about coloring the vertices of a few triangles. Astonishingly, it leads to a proof of the celebrated Brouwer fixed point theorem, which states that every continuous function $f : B^n \to B^n$ of an n-dimensional ball to itself has a fixed point.

An interesting section on tiling discusses Nicholas De Bruijn's result stating that if a rectangle R can be tiled by smaller rectangles each of which has a side of integer length, then R also has a side of integer length. A similar result is that a rectangle of side lengths a and b can be tiled with squares if and only if a/b is rational.

This is followed by three important results on finite sets. The first is by Sperner again, and it states that if F is a family of subsets of $[n] = \{1, 2, \cdots, n\}$ so that no element of F contains another element of F, then F consists of no more than $\binom{n}{n/2}$ sets. It is clear that that limit is attainable; just let F be the family of all $(n/2)$-element subsets of $[n]$. What is interesting is that this simple construction is the best. The situation is similar for the next problem as well. How many k-element subsets of $[n]$ can we choose if we want every pair of selected subsets to intersect? Clearly, $\binom{n-1}{k-1}$ is achievable (just choose all k-subsets containing 1). The Erdős-Ko-Rado theorem, presented here with the beautiful proof of Gyula Katona, shows that this is the best possible construction. Finally, we are shown the famous result of Philip Hall, which can be stated as follows. A school has a few

student clubs. Each club wants to send a representative to a meeting. No person can be the representative of more than one club. When is it possible to select a (distinct) representative from each club? The answer is that this is possible if and only if, for all k, the total membership of any k clubs is at least k.

Then we can read two sections on sophisticated enumeration methods, such as determinants to count lattice paths, and probabilistic counting techniques to figure out how many times we need to shuffle a deck of cards until the order of cards is truly random.

The next section contains four proofs for Cayley's formula for the number of trees on vertex set $[n]$. Their number is n^{n-2}, and mathematicians for over a century were fascinated by this fact. It has at least $4^{4-2} = 16$ proofs. The proof by André Joyal (by a superb bijection) and the proof by Jim Pitman (by double counting) are this reviewer's favorite proofs in this book.

The chapter closes by some examples of integer partition identities, and a discussion of which Latin rectangles can be completed to Latin squares.

6 Graph Theory

The part opens by Fred Galvin's solution to a question of Jeff Dinitz on coloring a particular graph. Let G be a graph whose vertices are the squares of an $n \times n$ grid. Two vertices are adjacent if and only if they are in the same row or column. Now let us associate a set of n colors to each vertex (the sets of colors associated to distinct vertices may or may not be distinct). The question is if we can color the vertices so that each vertex gets a color from the set that is associated to it, and adjacent vertices have different colors. The answer is positive, but it took fifteen years to prove that, even if the final proof is simple.

The graph theory part starts where the combinatorics part left off. We are shown a stronger version of the theorem that every map is five colorable, along the line of the Dinitz problem just discussed. Then comes a question from geometric graph theory, namely how many guards are needed to watch an art gallery that has n walls? The guards have to stay at given points, but can turn their heads. If the gallery is convex, then of course, one guard is enough. Without assuming convexity, $\lceil n/3 \rceil$ guards are always sufficient (and sometimes, necessary).

The next section is about Extremal Graph Theory. Let us assume that a simple graph G on n vertices does not contain a complete subgraph K_k. At most how many edges can G have? We are shown four different proofs for Turán's celebrated theorem stating that G can have at most $\frac{k-2}{k-1} \cdot \frac{n^2}{2}$ edges.

Then the authors use coding theory to motivate the following question. Let $\alpha(G)$ be the number of vertices in the largest empty subgraph of G. Let $\Theta(G) = \sup_{n \geq 1} \sqrt[n]{\alpha(G^n)}$, where G^2 is the direct product of G by itself. What is $\Theta(C_5)$, where C_5 is a cycle on five vertices? The classic result of Lovász is that $\Theta(C_5) = \sqrt{5}$. Two proofs are shown for this fact; both involve some linear algebra.

The next, recently added section reviews the history of Kneser's conjecture. Consider the graph $K(n, k)$ whose vertices are the k-element subsets of the n-element set, and in which the vertices corresponding to two subsets are adjacent if those two subsets are disjoint. The Kneser conjecture, from 1955 stated that the chromatic number of $K(2k + d, d)$ was $d + 2$. The authors explain the history of various proofs, and present the proof of Joshua Greene, which dates 2002.

The next-to-last section is on the *Friendship theorem*, a result that sounds innocent, but is not straightforward to prove. It states that if, in a company of at least three people, each pair of people has exactly one common friend, then there is a person there who is everybody's friend.

No book written to commemorate Erdős would be complete without his invention, the Probabilistic Method. This method proves that certain structures exist, without actually constructing them. Instead, it proves that a randomly selected object has less than 1 probability to *not* be the desired structure; hence it has a non-zero probability to be desired structure. Therefore, the desired structure exists. This idea is applied to prove a lower bound on diagonal Ramsey numbers. Then it is used to show that for each $k > 2$, there exists a graph whose smallest cycles are longer than k, and whose chromatic number is also larger than k.

7 Opinion

My experience is that most mathematicians have already looked at this book, and read the one or two sections that are closest to their field. Hopefully, this review will convince some of them to read the rest as well.

Review[11] of
Handbook of Chemoinformatics Algorithms
Edited by Faulon, Bender, eds.
CRC Press, 2010
440 pages, hardcover

Review by
Aaron Sterling (sterling@iastate.edu)
Department of Computer Science, Iowa State University
(visiting EECS Department, Northwestern University)

1 Introduction

This book is written by chemists, for chemists, with the objective of producing "an overview of some of the most common chemoinformatics algorithms in a single place." A first attempt at a definition of *chemoinformatics* might be, "Algorithms, databases and code to help chemists." Unlike the field of Bioinformatics, which enjoys a rich academic literature going back many years, the *Handbook of Chemoinformatics Algorithms* (HCA) is the first book of its kind. The editors (Faulon and Bender) commissioned authors to produce chapters on topics like computer-aided molecular design, open-source chemoinformatics software, and how to store chemical structures and properties in databases. Most of the algorithms presented, from the point of view of a theoretical computer scientist, are "folklore" or "trivial"; however, the application of these algorithms is complex, the problems computational chemists face are hard and not always well defined mathematically, and hundreds of millions of dollars ride on the speed and usefulness of practical implementations of these algorithms. Therefore, this book has the potential to be important to the TCS community: fundamental improvements to techniques in this volume could be a big deal.

That said, HCA is not for everyone. There are no theorems or proofs, most of the chapters are written in a laundry-list style ("This works in this case, and *this* works in *this* case"), and the language barrier is formidable. To provide a single data point, I took a year of chemistry in college, and did well, but even so, I found myself spending a lot of time reading Wikipedia articles just to understand what was being discussed. Further, the various authors do not appear to have experience framing things for a TCS reader. As one example, the phrase "non-polynomial (NP)-hard problem" appears in Chapter 9.

To aid in digestion of the material, I recommend one read the chapters out of order. HCA presents general technical details in earlier chapters, and motivating applications in later chapters. Specifically, I recommend the reader start with Chapter 13 (on sequence alignment algorithms) which is accessible with little chemistry background, then read Chapters 5-15 as desired. Refer to Chapters 1-4 to fill in gaps, or once the motivating applications are clear. (Readers with a background in Bioinformatics or computational biology may wish to start by reading Chapters 13-15, which apply techniques from those areas to chemoinformatics.)

Chemoinformatics itself is defined by the editors as "the field of handling chemical information electronically"; the term was coined by F. K. Brown in 1998. (Chemoinformatics is sometimes

 This work was supported in part by NSF grant CCF-1049899. The author would like to thank Bill Gasarch, who made helpful comments on an earlier draft of this review.

spelled "cheminformatics," without the first "o"; the terms are interchangeable.) According to Rajarshi Guha, who discusses open-source software in Chapter 12:

> Although the idea of free and shared software has been in existence since the 1960s, such software has not been generally available in chemoinformatics. In contrast, the field of Bioinformatics has produced a number of core algorithms (such as BLAST) as freely available software. Part of the reason for this is the fact that the majority of chemical information has been proprietary, in contrast to biological data (such as gene sequences and protein structures), which has traditionally been freely exchanged.

Both the editors and other chapter authors make similar points: because of the proprietary nature of chemical databases, etc., thoroughgoing academic (public sector) investigation into chemoinformatics is a recent phenomenon, and many functionalities are not available for free, nor inexpensively—nor, publicly, at all.

As a general warning to the reader, I was unable to determine how well the techniques in HCA work, even after reviewing several Wikipedia articles and one other computational chemistry book. Authors of most chapters in HCA make the point that the techniques they present have been used to find compounds of potential interest more inexpensively than by using wetlab experimentation alone, but the ratio of hits to misses is not something I was able to track down. It does seem, especially in the area of QSAR models that I discuss in Section 2, that there has been serious criticism leveled against *in silico* research as publications devoid of real-world applicability. Some authors in HCA address those concerns, and present techniques with safeguards they claim solve those problems, but, even so, they provide only anecdotes of success, no hit-and-miss ratio. Still, the algorithms in HCA are employed by petroleum companies, chemical companies, pharmaceutical companies, universities worldwide, and the FDA, so, for better or for worse, they are what's currently available.

I will focus this review on the two chemoinformatics applications most discussed in HCA: quantitative structure-activity relationships (QSAR), and computer-assisted molecular design (CAMD). Then I will briefly consider the contents of HCA chapter by chapter.

2 Principal motivating applications

2.1 Quantitative Structure-Activity Relationships (QSAR)

QSAR is far and away the most-discussed application in HCA; it is mentioned in every chapter. A *quantitative structure-activity relationship* is a mathematical relationship between a molecule's physical properties and its chemical properties, along the lines of, "All molecules whose molecular graphs contain subgraph G will produce reactant R when reacting with chemical C." The primary purpose of QSAR is to *make predictions* about how an as-yet-unstudied molecule will behave, based on better-known molecules that are structurally similar to it. Some QSAR approaches use algorithms that solve the Graph Isomorphism Problem; while this can work for small examples, or in situations where heuristics help, all known Graph Isomorphism algorithms are (of course) worst-case infeasible. Therefore, the main QSAR approaches collapse the information in molecules down to individual real numbers called *molecular descriptors*, which are the topic of Chapter 4. For different applications, many different molecular descriptors may be required.

The Wikipedia article on QSAR is well-written, and I recommend it. Tropsha and Golbraikh in Chapter 6 of HCA describe QSAR as follows.

> The discovery of novel bioactive chemical entities is the primary goal of computational drug discovery, and the development of validated and predictive QSAR models is critical to achieve this goal.... The main QSAR hypothesis underlying all QSAR studies is as follows: similar compounds should have similar biological activities or properties. If this condition for compounds in the dataset is not satisfied, building truly predictive QSAR models is impossible. In fact, one can define two compounds to be similar if their chemical structures are similar. In computer representation, compounds are characterized by a set of quantitative parameters called descriptors. *Similarity* between two compounds is a quantitative measure that is defined based on compounds' descriptor values. Different definitions of compound similarity exist.... Obviously, quantitative values of similarity measures between two compounds also depend on which descriptors are used. So there is no unique similarity measure.

QSAR is more complicated than "just" a massive chemical database search, because, in reality, structurally similar molecules can have very different properties. (This is sometimes termed the "SAR Paradox.") As a result, much of the literature on *in silico* QSAR provides only *conditional results*, along the lines of: assuming molecule family $\mathcal{F}$ has similar properties to its structural cousins $\mathcal{F}'$, we can deduce results X, Y and Z. Recently, prominent scientists and scientific organizations have called much of the QSAR research into question, because of the uncertainty (and probable falsity) of many starting assumptions. The European Organization for Economic Co-operation and Development now requires "appropriate measures of goodness-of-fit, robustness and predictivity" for QSAR models. To address these concerns, HCA devotes two chapters to QSAR: Chapter 6, on data preparation and modeling; and Chapter 7, on development and validation of the QSAR models themselves.

2.1.1 Chapter 6: Predictive Quantitative Structure-Activity Relationships Modeling: *Data Preparation and the General Modeling Workflow*

There are now large (millions of compounds) chemical databases publicly available online. In Chapter 6, the authors (Tropsha and Golbraikh) present their workflow for developing predictive QSAR models from such databases. The workflow breaks down to the following general steps: data treatment and preparation, calculation of molecular descriptors (i.e., construction of a similarity metric over the dataspace); preprocessing those descriptors; clustering the dataspace according to the similarity metric; and detecting and removing outliers from those clusters (i.e., compounds that do not share properties with other cluster elements, even though they are metrically close to them). A cluster thus produced can serve as a QSAR model. Finally, in addition to model validation, the authors recommend laboratory verification of the *in silico* prediction that compound X will have property Y.

The authors use a machine-learning approach. From the initial dataset, they first remove 10-20% of the compounds at random, for use as an external evaluation set, for eventual model verification. The remaining compounds are divided into multiple training and test sets. The current computational upper limit of the size of the training set is 2000 compounds. The authors recommend a minimum of 20 compounds in the training set, and a minimum of ten compounds in each of the test and external evaluation sets, to avoid chance correlation and overfitting.

Once compounds are chosen and assigned to sets, descriptors are calculated for each molecule. Molecular descriptors are real numbers that quantify a molecular property. Examples that do not require chemistry vocabulary include: how many different molecular groups (i.e., labeled subgraphs appearing in a set of subgraphs) appear in a given molecule; or, how many distinct spanning trees occur in a molecular graph. Over 2000 molecular descriptors appear in the chemoinformatics literature, and this calculation step is complicated by economics: most chemical descriptors are available only from (expensive) commercial software, although the FDA has recently produced free software that generates some descriptors.

Once the descriptors are calculated, the authors build a multidimensional descriptor space—and finally we can talk about theoretical computer science! The multidimensional descriptor space is a metric space with each axis defined by a different descriptor. The most common metrics used on this space are the Euclidean distance, the Mahalanobis distance (which weights correlation between descriptors), and the Tanimoto coefficient, which I will present here because it is "the most popular similarity measure used in chemoinformatics," according to Clark and Roe, the authors of Chapter 5. Given two vectors $\overrightarrow{x_1} = \langle x_{10} x_{11} \cdots x_{1n} \rangle$ and $\overrightarrow{x_2} = \langle x_{20} x_{21} \cdots x_{2n} \rangle$, the Tanimoto similarity measure (coefficient) S_{TAN} is defined as:

$$S_{TAN} = \frac{\sum x_{1j} x_{2j}}{\sum x_{1j}^2 + \sum x_{2j}^2 - \sum x_{1j} x_{2j}}.$$

Recall that the dot product of two vectors divided by their geometric mean is the cosine of the angle between those vectors, which provides a measure of how far the vectors are from each other. The Tanimoto coefficient is the dot product of two vectors divided by their arithmetic mean with a correction for the size of the dot product. So it is like the cosine measure, but it provides more resolution when measuring similarity between vectors that are nearly maximum length in the descriptor space.

Tropsha and Golbraikh devote several pages to normalization of descriptors, which I will skip. Once the metric space is defined and the axes normalized, it is time to analyze clusters of data points. Computation of the distance between each point may be infeasible, so the authors present the following randomized algorithm. Decide upon a minimum similarity threshold. Then choose an initial compound at random and put it into the cluster. Choose another compound at random and put it into the cluster if its similarity with every cluster element exceeds the threshold. (Otherwise, discard it.) Continue until all compounds have been considered. (This concludes the algorithm.) The authors suggest running this algorithm repeatedly on the same compounds, with significantly different thresholds, until finding a threshold that selects a cluster with the number of elements desired.

Once the cluster space is mapped, the next step is to remove outliers, so all points in a given cluster do indeed share common molecular properties. A "leverage outlier" is a point distant from all other points, even though it may lie at the geographic center of the distribution. The authors present two methods to remove leverage outliers. Once the leverage outliers are removed, one must remove "activity outliers," which are points that may be close to other points in the descriptor space, but whose chemical activity is nonetheless quite different. This step seems to rely more on chemical intuition than CS theory.

A cluster thus prepared can function as a QSAR model, for, e.g., *in silico* pharmaceutical or solvent design. To establish and validate the predictive power of such a model, the authors advocate a machine-learning procedure, "during which the model is 'trained' (i.e., model parameters are

tuned to provide the highest predictivity in terms of some statistical criterion used as a target function which is optimized during the procedure)." This is the topic of Chapter 7, which I will now briefly consider.

2.1.2 Chapter 7: Predictive Quantitative Structure-Activity Relationships Modeling: *Development and Validation of QSAR Models*

The authors of Chapter 7 (Tropsha and Golbraikh again) measure the correctness of a predictive QSAR model by running statistical tests on target functions the model is trained for. One such target function is Correct Classification Rate (CCR), defined by $CCR = N(\text{correct})/N(\text{total})$, where $N(\text{correct})$ is the number of compounds the model has classified correctly, and $N(\text{total})$ is the total number of compounds considered. Most of the measures of the target function are well-known statistical tests, like leave-one-out cross-validation.

Rather than just test a particular descriptor space that is optimized in a particular way, the authors recommend the combinatorial QSAR approach (combi-QSAR). In combi-QSAR, one starts with a single dataset and produces several different descriptor spaces, with different collections of descriptors. Then take each descriptor space, and consider its behavior under different optimization methods. Then choose the best models out of all these, and take the consensus predictions of those best models. The authors claim this approach adequately address the criticisms leveled against the QSAR literature, and will allow QSAR to be a significant tool for drug exploration and discovery in the 21st century.

2.2 Computer-Aided Molecular Design

Other than QSAR, the principal application considered in HCA is Computer-Aided Molecular Design (CAMD). Visco, the author of Chapter 9, explains CAMD as follows.

> CAMD is the application of computer-implemented algorithms that are utilized to design a molecule for a particular application.... [W]hen the desired molecules are for biological systems, the solution space is estimated to be at least 10^{60}, which is a relatively tiny fraction of the space as a whole.... Accordingly, efforts are made *a priori* to limit the search space using techniques such as full-fledged templating or requiring the presence of certain features in a candidate molecule.
>
> The two most visible industries using CAMD are the chemical process industry and the pharmaceutical industry. While both industries are solving CAMD problems, they differ in substantial ways. For example, the chemical process industry regularly uses CAMD in the area of solvent design. Solvents are designed to have certain properties for applications in a particular area and outputs of CAMD algorithms are scored based on predicted properties.... In the pharmaceutical industry, CAMD is often used in a *de novo* approach.

Chapter 9 focuses on molecules designed using QSAR. Chapter 10 focuses on molecules designed *de novo* (Latin for "from the beginning"), i.e., building compounds from scratch to fit a target receptor.

2.2.1 Chapter 9: Computer-Aided Molecular Design: *Inverse Design*

Historically, the first CAMD algorithms combined functional groups (predefined molecular subgraphs) to generate a larger structure. This suffered from combinatorial explosion, as the algorithms produced many nonfeasible structures. A more feasible approach, *inverse QSAR* (iQSAR) began to be studied in the early 1990s. In iQSAR, "rather than using a QSAR to score potential molecules created from combining groups, one fixes a desired value (or range of values) and attempts to solve for the set of independent variables (descriptors) that satisfy the QSAR." The first step in this general approach is to determine the pool of molecular fragments that will be used to build the lager structures. Then one builds candidate structures and evaluates their fitness.

Visco, the author of Chapter 9, presents several algorithms used in Hybrid-CAMD, a popular approach which "combines the generate and test paradigm of previous techniques with inclusion of higher-order information (such as molecular connectivity through topological indices)."

The author then considers CAMD as an optimization problem. This approach minimizes the differences between the desired property and the prediction of the candidate structure's possession of that property. Visco presents a Mixed Integer Linear Programming (MILP) algorithm for CAMD, and discusses the application of MILP to designing soybean oil products and a new pharmaceutical with a penicillin backbone. Finally, Visco considers *CAMD using signature*, a method that structures molecular generation by providing a database of atomic "signatures" and a molecular "signature" that is the sum of the atomic signatures that appear in that molecule. Producing a molecule of interest can then be done by solving a set of Diophantine constraint equations that provide a minimum and a maximum for each atomic signature allowed to appear in the molecule one wishes to design.

2.2.2 Chapter 10: Computer-Aided Molecular Design: *de novo design*

Roe, the author of Chapter 10, begins with an high-level explanation of *de novo* design, and its advantages and drawbacks with respect to iQSAR.

> PubChem, the largest database of known chemicals, has a little more than 19 million compounds to date.... Even combinatorial libraries, which can range up in size to billions of compounds, do not begin to fully sample the range of all possible compounds. As the full compound space is too vast to search comprehensively, strategies have to be employed to search this space efficiently for the discovery of novel lead drug compounds.
>
> *De novo* design handles this challenge by building compounds from scratch to complement the target receptor. The guiding principle in this approach is that small molecules that are complementary to the target receptor, both in shape and chemical properties, will have the most specific binding.... To reduce the search of chemical space to a manageable problem, strong physical constraints must be taken into account at each step during the generation of the lead drug molecule, limiting the chemical space explored to those regions specifically complementary to the target receptor. The advantage of this approach over a virtual screening strategy [i.e., QSAR-style strategy] to identify these compounds is that the search space is directed to the relevant regions in chemical space with a far greater range and diversity of potential lead compounds that can be evaluated.... The main drawback is that the resulting compounds need to be experimentally synthesized and tested, rather than taken from an in-house inventory or ordered

commercially.

Roe presents three CAMD algorithms—Grow, Fragment-Link, and Evolutionary Strategies based on sampling—and discusses programs that currently run each. Grow has performed well when all features of a receptor are near one another, while Fragment-Link has performed better if the receptor consists of two or more subpockets separated by a large gap. Sampling strategies do not direct molecular generation explicitly toward linking with interaction sites, but rather they maximize a general fitness evaluation, so they tend to produce structures that are more stable (have lower molecular energy). However, sampling strategies have to investigate much larger search spaces than do either Grow or Fragment-Link.

Grow is a Metropolis Monte Carlo algorithm. At a given stage, a candidate to add to the structure is chosen at random and its probable binding affinity is calculated. Then a random number is generated, and if the binding affinity is greater than that random number, the candidate is added to the structure; otherwise, it is discarded and the process repeats. Fragment-Link requires more user interaction. The algorithm places binding structures in all the "hot spots" of the receptor, and then a user has to clean up the molecule through visual inspection. Evolutionary strategies are often programmed as genetic algorithms; Roe presents pseudocode for one such strategy.

3 Summary of Chapters

Chapter 1 (by Ivanciuc) discusses molecular graphs. It begins by introducing notions familiar to many SIGACT News readers (walks, adjacency matrix, Laplacian matrix, etc.), and then presents many different ways to represent two-dimensional chemical structures as graphs. For example, one can weight both vertices and edges, as well as labeling vertices with chemical symbols. Vertices then represent atoms (or, more generally, chemical groups); and edges, the bonds between those atoms. When constructing a graph for a biological (i.e., carbon-based) molecule, the vertices are weighted according to the atomic number of the atom the vertex represents, normalized so the weight of a carbon atom is 0. An edge representing a single bond is weighted 1; one representing a double bond, 2; etc.

Chapter 2 (by Misra and Faulon) considers **Algorithms to Store and Retrieve Two-Dimensional (2D) Chemical Structures**. One of the most-used approaches to build chemical databases is SMILES (Simplified Molecular Input Line Entry Specification), which produces a linear string representation of a molecule. The SMILES representation can then be parsed using string processing algorithms. (For example, the SMILES string for acetaminophen is "C(=O)[Nc1ccc(cc1)O]C".) The authors present the canonical SMILES algorithm: it has a worst-case running time of $\mathcal{O}(n!)$ if one wants to find the minimum-length SMILES string that correctly represents a molecule. The authors discuss more recently proposed notations that do not have this problem. Also, as one can see from the string for acetaminophen, SMILES only records names of molecules and their qualitative relationships. It is useful to include more information in a database, like the physical distance between pairs of atoms. To do this, more complex languages have recently come into use, like CML (Chemical Markup Language), an XML-based molecular file format hosted on Sourceforge.

Chapter 3 (by Willighagen) is titled **Three-Dimensional (3D) Molecular Representations**. The author's main concern is "how to represent 3D molecular geometries such that they are useful for analysis and computation," because a coordinate representation useful for visualization may be cumbersome tool for data analysis or pattern recognition. Therefore, much of the chapter

examines how to convert between different coordinate systems, and how to produce a numerical, fixed-length vector representation (i.e., a molecular descriptor) from a molecule's coordinate representation. Willighagen presents an algorithm to calculate the molecular center-of-mass from a Cartesian representation of the molecule; from a TCS point of view, this calculation proceeds in the "expected" way. He defines a notion of "internal coordinates," which provide a 3D description of a molecule without an external frame of reference, by describing "the molecular geometry in terms of distances between atoms and angles and torsions between bonds." He presents an algorithm to convert internal coordinates to Cartesian coordinates. Willighagen also discusses how to compare two 3D geometries: one method is to find the maximum common substructure (MCSS) shared by two molecules, and to orient the molecules by minimizing the root mean square deviation of the coordinates of their shared MCSS.

Several pages of Chapter 3 are devoted to a practical example of the use of Radial Distribution Functions (RDFs) to classify different types of crystal packing, so I'll make the following advisement to the reader. The Wikipedia article on RDFs states that RDFs are a measure of how atomic density is distributed through a molecule. However, RDFs are used in HCA in a more general way: to measure the geographic distribution of some feature in the molecule, or to evaluate how influential an atom is on its neighbors and neighbors' neighbors, etc.

Chapter 4, **Molecular Descriptors** (by Fechner, Hinselmann and Wegner), is the largest chapter in HCA, at over fifty pages. It presents multiple ways to measure structural properties with real numbers, from graph-theoretic topological indices, to hash functions on molecular graphs, to notions of edit distance, to graph kernels and kernel functions. It also presents pseudocode for several algorithms, but many of these are "introductory" from a TCS perspective, such as Dijkstra's shortest path, or DFS/BFS traversal. More interesting is the algorithm for shapelet extraction, which approximates the shape of a molecule by local approximations to its surface using hyperbolic paraboloids. (The definition of a molecule's surface is not something I will get into here.) Speaking philosophically, the authors note:

> Graph complexity can be defined in various ways, but still there is no standard definition. In [*Complexity and Chemistry: Introduction and Fundamentals* by Bonchev and Rouvray] various criteria are compiled from different sources, which describe the requirements for a "good" molecular complexity descriptor. For example, a complexity index should
>
> - Increase with the number of vertices and edges
> - Reflect the degree of connectedness
> - Be independent from the nature of the system
> - Differentiate nonisomorphic systems
> - Increase with the size of the graph, branching, cyclicity, and number of multiple edges.
>
> Still, this is an ongoing discussion, with even conflicting positions.

Rather than attempt to summarize the breadth of material in Chapter 4, I will offer an opinion. The material is presented in a style of, "This descriptor has worked for some applications, now let's move on to the next descriptor." The approach is pragmatic, and even in the more specialized *Complexity and Chemistry*, there is no construction of a mathematical theory, only informal

arguments from induction on small examples. Also, it seems, from a literature search, and two questions I asked on cstheory.stackexchange.com, that these molecular descriptors have not been studied much in TCS or in graph theory (though more general notions certainly have). Therefore, I believe there is an interdisciplinary research opportunity here.

Chapter 5 (by Clark and Roe) is on **Ligand- and Structure-Based Virtual Screening**. The authors define virtual screening as "the selection of molecules likely to show desired bioactivity from a large database." In other words, once one has assigned descriptors to molecules, the objective is to select molecules that are similar to one another. Clark contributes a section on similarity searching, in which he discusses similarity metrics, including the Tanimoto coefficient.

Roe begins by considering the *docking problem*: given a target receptor, find the molecule that docks best in that receptor (i.e., find the best-fitting peg for a hole). As Roe puts it:

> The docking problem can be broken down into three components: (1) orientational search, or the search for the 3D orientation of a molecule with respect to another; (2) conformational search, or the search through rotatable torsions; and (3) scoring, or evaluation "pose" or orientation/conformation by some measure of predicted binding.

Docking algorithms include the search for all cliques of a limited size (this search problem is tractable), in order to find sections of the receptor that match up correctly to sections of the molecule (this is an example of orientational search); and the Lamarckian Genetic Algorithm (LGA), which combines orientational and conformational search. In LGA, the genetic algorithm's chromosome is created to represent the orientation/conformation of the molecule with respect to the target receptor. This chromosome then evolves according to a fitness function. LGA is nondeterministic, and it terminates based on user parameters.

Chapters 6 and 7 are about QSAR modeling, and I have considered them already in Section 2.

Chapter 8 (by Meringer) is on **Structure Enumeration and Sampling**. The motivating questions are:

> Given a molecular formula plus, optionally, a list of structural constraints, the typical questions are: (1) How many isomers [nonisomorphic graphs satisfying these constraints] exist? (2) Which are they? And, especially if (2) cannot be answered completely: (3) How to get a sample?

The mathematical foundation used to answer these questions is Pólya's theory of counting. Meringer gives an example of Pólya's theory by counting the 22 permutational isomers of dioxin by calculating the cycle index of the binding sites of the molecular graph and then inserting a generating function into the cycle index. Meringer generalizes this approach to an algorithm and provides more examples. The author then considers the construction problem of producing all structurally distinct molecules that have the same chemical formula (i.e., all distinct isomers). He begins with the DENDRAL system, which was perhaps the first expert system of any sort, and was "one of the roots of chemoinformatics." DENDRAL (short for DENDRIC ALgorithm) was generating isomers as early as the 1960s.

Until the 1970s, only acyclic structures could be constructed. Eventually, Masinter described a cyclic structure generator, which Meringer sketches. The weakness of Masinter's generator is an inability to process structural constraints effectively, so different methods were introduced that, for example, put an ordering on the set of graphs to be generated, so it is not necessary to do

pairwise isomorphism testing. One can accelerate structure enumeration by placing constraints (like planarity, or minimum cycle size) on the graphs to be generated. The author mentions several other generation methods. Nevertheless, "it is often impossible to generate all molecular graphs belonging to a given molecular formula" due to combinatorial explosion. As a result, sampling techniques are essential, in order to produce a representative subset of the total isomer space. Monte Carlo methods, simulated annealing and genetic algorithms are most often used for this purpose. Meringer presents pseudocode for a simulated annealing algorithm and a genetic algorithm for isomer sampling. The chapter concludes with the application of constructing *combinatorial libraries*, i.e., chemical databases where either the compounds possess high structural diversity, or, alternatively, all share some common relation.

Chapters 9 and 10 are about computer-aided molecular design, and we have considered them already in Section 2.

Chapter 11 (by Faulon and Carbonell) shifts gears, and instead of focusing on molecules, it focuses on **Reaction Network Generation**. The authors explain:

> Designing new drugs or chemicals, understanding the kinetics of combustion and petroleum refining, studying the dynamics of metabolic networks, and using metabolism to synthesize compounds in microorganisms are applications that require us to generate reaction networks. As the networks may be quite large, there is a need to automate the procedure....
>
> While formal techniques are robust, their computational scaling limits their applicability to real reacting systems....[T]he number of reactions and intermediate species that are created generally scales exponentially with the number of atoms of the reactants.... With all these systems, not only the time taken to generate networks may scale exponentially, but simulating the dynamics of large networks may be computationally prohibitive.

Note that the simulation cost is not exponential, but "only" N^3 where N (the number of chemical species) can easily be larger than 10^4 or 10^5.

The authors present a network generation/sampling algorithm, which simultaneously generates a reaction network, and also "reduces" it, i.e., removes reactions not critical to the particular task at hand. The method of applying a reduction strategy is a standard one to mitigate the otherwise exponential blowup of the reaction network, and the definition of a reduction strategy will depend on what products of a reaction are considered important. The authors first present pseudocode for a deterministic network generator, and illustrate its application to ethane thermal cracking. There is a comprehensive database of 324 elementary transformations that carbon atoms can take in organic reactions. The input of the deterministic reaction generator is "a list of reactant species, a list of constraints, and a list of elementary transformations." The authors then present pseudocode for three distinct sampling algorithms that are stochastic, not deterministic, and "samples" the total set of reactions generated at each stage, preserving only the ones considered most important. One way of measuring importance, for example, is to keep only the reactions whose reactants have the highest concentration in the system.

Chapter 12 (by Guha) covers **Open Source Chemoinformatics Software and Database Technologies**. These open-source efforts are quite recent (all starting in the 2000s). For example, the SMILES language (which as we saw above produces strings that represent chemicals in databases) has suffered from ambiguities in its original specification, and proprietary implementations of SMILES have extended it in different ways. Therefore, there is now an open-source

OpenSMILES project which intends "to explicitly define the SMILES language in a public manner. The end result is expected to be a formal grammar for the language along with reference implementations" that is completely free-of-charge. Guha compares three open source chemoinformatics toolkits (CDK, OpenBabel, and RDKit), reviewing which functionalities they provide and their development activity. All three toolkits are hosted on SourceForge. A computer scientist could read this chapter with little chemistry background.

As I mentioned in Section 1, Chapters 13-15 are applications of Bioinformatics techniques to chemoinformatics. Indeed, **Chapter 13** (by Akutsu) on **Sequence Alignment Algorithms: *Applications to Glycans and Trees and Tree-Like Structures*** could be considered pure Bioinformatics. Glycans are carbohydrate sugar chains that are usually found on the surface of cells. According to Akutsu, "The importance of Glycans has been recognized in the field of Bioinformatics since the beginning of the twenty-first century and then various studies have been carried out." There is now a publicly available Glycans database called KEGG Glycan, and a search tool with alignment algorithms for that database, called KCaM (KEGG Carbohydrate Matcher). Like RNA or DNA, the alignment problem for Glycans is: given two Glycans, find the way to align them so there is maximum possible matching of locations. Glycans are usually represented as rooted trees, so the main theoretical tool used is tree edit distance (i.e., how many steps it takes to turn one tree into another by using only node deletions, insertions and substitutions). Akutsu presents one tree-edit distance algorithm and discusses others. The author then considers Glycans as unrooted trees. Tree-edit distance becomes NP-hard in that case, so KCam algorithms have been developed that combine finding the Maximum Common Subtree (MCST, which can be done in polynomial time) with the standard Bioinformatics tools of score matrices and local alignment. Akutsu discusses an MCST algorithm and presents pseudocode for both global and local Glycan alignment algorithms.

Chapter 14 (by Martin) is titled **Machine Learning-Based Bionformatics Algorithms: *Applications to Chemicals***. The author first considers applications of clustering to Bioinformatics and chemoinformatics, and then considers applications of classification and regression to both fields. Martin presents pseudocode for an agglomerative hierarchical clustering algorithm , and for the Jarvis-Patrick clustering algorithm. (Hierarchical clustering is $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space, while Jarvis-Patrick is $\mathcal{O}(n^2)$ time but only $\mathcal{O}(n)$ space, so Jarvis-Patrick is the preferred method used in chemoinformatics.) This chapter is readily accessible to most TCS readers, and, for purposes of this review, I will focus on the author's description of the differences between the two fields.

> [B]ioinformatics uses a much wider variety of algorithms, but...chemoinformatics uses a much broader set of data types. Bioinformatics applications use a huge variety of algorithms but are generally restricted to microarray (numerical) and sequence (string) data. Chemoinformatics applications generally use two to three clustering algorithms (Jarvis-Patrick, Ward clustering, and k-means) but use very large datasets and a huge number of chemical descriptors as their input....
>
> In [the QSAR] approach, the...protein is usually fixed and all attention is focused on the drug. In Bioinformatics, it is more common to consider a network of interactions, as in the case of a protein-protein interaction network. If we combine both types of approaches, we arrive at the prospect of predicting a protein-chemical interaction network. (Although the general area of large-scale (informatics-based) protein-chemical interac-

tion prediction is relatively unexplored, both approaches mentioned here use SVMs.)

HCA concludes with **Chapter 15** (by Mu, Bauer, Faeder and Hlavacek) on **Using Systems Biology Techniques to Determine Metabolic Fluxes and Metabolite Pool Sizes**. This chapter begins with a discussion of Flux Balance Analysis (FBA). Intuitively, suppose you know the individual reactions that take place between a cell and its exterior, and you have measured experimentally how the mass of the cell changes over time. Then, by plugging all the reactions into a "stoichiometric matrix" and setting the objective function as the (already known) mass of the cell, it is possible to determine how the reactions affect one another—to understand the relations among the cell's different biological pathways. FBA is a computational technique. An experimental technique related to FBA is to mark certain molecules with radioactive isotopes (often ^{13}C). Then, by measuring the labeling patterns created by allowing reactions to progress, one can determine the fluxes (i.e., types of interrelations) of the reactions using the isotopes. Most of this chapter is devoted to ^{13}C Metabolic Flux Analysis (^{13}C MFA). In the words of the authors, "^{13}C MFA has reached a state of relative maturity. The experiments themselves have become a routine procedure, the measurement techniques are well-established, and sophisticated mathematical evaluation algorithms are available. However, ^{13}C MFA is still a time-consuming and low-throughput process." Therefore, Chapter 15 focuses on *in silico* ^{13}C MFA. By using a database of "carbon fate maps" that simulate how carbon isotopes move within individual reactions, it is possible to program a complex system in the BioNetGen software package, and graph the results using MATLAB. The authors devote several figures to BioNetGen code for a specific example.

4 Conclusion

While *Handbook of Chemoinformatics Algorithms* is probably a must-have for computational chemists, it will only interest computer scientists who are actively looking for an interdisciplinary project, and who are willing to put in the effort required to learn the vocabulary and perspective of a different field. This review has been heavy on application-explanation but light on math, and HCA has the same flavor. I wrote a much longer review than "necessary," in an attempt to build a bridge between chemoinformatics and TCS, and I hope this bridge encourages more computer scientists to investigate this area. Chemists could use help from TCS to build a mathematical and algorithmic theory for their work, and I believe chemoinformatics, like Bioinformatics, will prove to be an important source of problems for computer science. My suspicion is that a "small" increase in the mathematical sophistication of one or more of the algorithms in HCA could yield a big payoff.

Review[12] **of**
Dynamic Fuzzy Logic and its Applications
by Fanzhang
Nova Science Publishers, Inc., New York 2008
x+296 pages, hardcover

Reviewed by Song Yan syan@math.harvard.edu

1 Introduction

According to Zadeh [4], a fuzzy set is a class of objects with a continuum of grades of membership. Such a set is characterized by a membership function which assigns to each object a grade of membership ranging between 0 and 1. Hence, the theories of fuzzy sets and fuzzy logic, provide a mathematical tool suitable for reasoning about uncertainty in static environment. The dynamic fuzzy sets and dynamic fuzzy logic (see [1], [2] and [3]), however, are natural extensions to static fuzzy sets and static fuzzy logic for reasoning uncertain information in dynamic environment. Although a large amount of work has been done in fuzzy logic and fuzzy sets, little has been done in dynamic fuzzy sets and dynamic fuzzy logic, particularly in a book form. The book *Dynamic Fuzzy Logic and Its Applications* by Fanzhang Li fills the gap.

2 Overview

The book starts by a Foreword by Yi Pan, Chair and Professor in the Department of Computer Science at Georgia State University and a Preface by the author. The Foreword and the Preface provide a brief introduction to the book. The main contents consist of 12 chapters on the two main topics: dynamic fuzzy sets and dynamic fuzzy logic. These 12 chapters can be naturally divided into 3 parts. The first part, consisting of chapters 1 to 6, introduces the basic concepts and results of dynamic fuzzy sets. The second part, consisting of chapters 7 to 11, discuses the basic concepts and results of dynamic fuzzy logic. The last part, i.e., Chapter 12, discusses the applications of dynamic fuzzy logic to machine learning and reasoning. More specifically, Chapter 1 introduces the basic concepts, ideas and operations of classical fuzzy sets, these concepts are then generalized to dynamic fuzzy sets. For transformation between dynamic fuzzy sets and classical fuzzy sets, the concept and properties of cut sets are introduced. Chapter 2 deals with the module operations, the axiomatic structure of $\wedge$ and $\vee$, the decomposition theorem of dynamic fuzzy sets, respectively. Chapter 3 studies the lattices properties and module structures of dynamic fuzzy sets. Chapter 4 presents the basic definitions and results, particularly the representation theorem of nested fuzzy dynamic sets. Chapter 5 gives discusses the extension theorem of dynamic fuzzy sets, the probability and operation, and particularly the measure space theory of the dynamic fuzzy number. Chapter 6, the last chapter of the first part, deals mainly with the dynamic fuzzy measure theory, including dynamic fuzzy relation, dynamic fuzzy relation equation, dynamic fuzzy measure, and solution to the dynamic fuzzy integral. Chapter 7, the first chapter of part 2 on dynamic fuzzy logic, is on fuzzy Boolean variables, propositional calculus, predicate calculus and resolution principle of

dynamic fuzzy logic, which is a natural extension of the classical fuzzy logic to dynamic fuzzy logic. Chapter 8 discusses automated reasoning methods within the framework of dynamic fuzzy logic, including dynamic fuzzy knowledge representation and dynamic fuzzy reasoning rule description. It also contains discussions on connection reasoning and default assumption reasoning of dynamic fuzzy logic and their applications in medical diagnosis. Chapter 9 provides a basis for dynamic fuzzy logic programming languages (including syntax and structures operational semantics), data types, Lambda calculus,verifications and correctness proof in dynamic fuzzy logic. Chapter 10 presents a multi-agent learning model based on dynamic fuzzy logic. This model consists of agent intelligence model, agent intelligence states and their axioms, agent working theory, single-agent learning algorithms, and multi-agent learning algorithms. Chapter 11 presents an autonomic computing model based on dynamic fuzzy logic, which can be used in many real-world applications including business decision-making. Chapter 12, the last chapter of the book, discusses applications of dynamic fuzzy logic to machine learning, particularly, it studies a geometric model of dynamic fuzzy machine learning. The book finally ends with 397 bibliographic items and about 300 index entries.

3 Opinion

Although fuzzy sets and fuzzy logic provide tools for reasoning about uncertain information, it seems that they are unable to provide solutions to dynamic problems. This book, however, offers alternative tools for reasoning about dynamic uncertain information. It extends the static fuzzy sets and fuzzy logic to the dynamic fuzzy environment and provides tools in machine learning for dynamic fuzzy systems. Although the book is basically a collection of results published in paper form, jointly by the author and his graduate students, it is largely self-contained; it is easy to read and easy to follow. It is suited either as a basic reference in the field, or as graduate text in the relevant courses in fuzzy mathematics and machine learning.

Reference

[1] J. X. Lee and G. Vukovich, The Dynamic Fuzzy Logic System: Nonlinear System Identification and Application to Robotic Manipulators, *Journal of Robotic Systems*, Vol 14, 1997, pp 391405.

[2] O. V. R. Murthy, R. K. P. Bhatta and N. Ahmada, Extended Dynamic Fuzzy Logic System (DFLS) Based Indirect Stable Adaptive Control of Non-Linear Systems, *Applied Soft Computing*, Vol 4, 2004, pp 109-119.

[3] J. L. Pérez-Silva and F. Lara-Rosano, Dynamic Fuzzy Logic, *MICAI 2000: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, Vol 1793, Springer, 2000, pp 661-670.

[4] L Zadeh, Fuzzy Sets, *Information and Control*, Vol 8, 1965, pp 338-352.