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. Joint review of **How to Solve It: A New Aspect of Mathematical Method** by Polya and the new **Street-Fighting Mathematics** by Mahajan. Review by Mohsen Vakilian. Both books are about how to solve problems. Polya's is a classic and stresses rigor, Mahajan's book is more concerned with getting the right answer. Not that there's anything wrong with that.

2. **Grammatical inference: learning automata and grammars** by Colin de la Higuera. Review by Matthias Gallé. How do you build a grammar or an automaton representing a language of which you only have partial information? This leads to many many question. This book summarizes much of what is known about this question.

3. **Logical Foundation of Proof Complexity** by Stephen Cook and Phuong Nguyen. Review by Arthur MILCHIOR. If A is a decision problem in P, how can you express A as a formula? How complicated would the formula have to be? How complex would a proof of $x \in A$ have to be? These are examples of questions from Proof Complexity, the topic of this book.

4. **Exact Exponential Algorithms** by Fedor V. Fomin and Dieter Kratsch. Review by Michael Lampis. Lets say a problem A is NP-complete and the best algorithm for it runs in 2^n time. Oh well. WAIT A MINUTE- maybe we can do better. We can't do MUCH better but, hey, perhaps we can do the problem in $(1.8)^n$ time! This book is about exponential algorithms that get the correct answer and looks at how to make them practical by lowering the exponent.

5. **Bioinspired Computation in Combinatorial Optimization** by Frank Neumann and Carsten Witt. Review by Sven Herrmann. There are some algorithms for problems in combinatorial optimization that use ideas from biology. They are mostly stochastic search algorithms inspired by evolution, though there are other ways as well. The material on this is now gathered together in one book!

6. **Triangulations: Structure for Algorithms and Applications** by Jesús A. De Lorea, Jörg Rambau, and Francisco Santos. Review by Michael Murphy. The review says that *This book is about triangulations of a fixed point set in Euclidean space of arbitrary dimension. The authors focus particularly on flip-graph connectivity and regular triangulations, but they also cover applications to algorithmic and complexity questions, as well as techniques relating triangulations to convex polytope theory. The many illustrations and nice exposition make the book a joy to read for people interested in Computational Geometry, Linear Programming, and Convex Polytopes.*

7. **Flows in Networks** by L. R. Ford Jr. and D. R. Fulkerson. Review by Yixin Cao. This is the classic book on Network Flows. The Network flow problem is an important problem which has been at the heart of algorithms for many years. And here is where it all started!

8. **Quantum Computing - A Gentle Introduction** by Eleanor Rieffel and Wolfgang Polak. Review by Kyriakos N. Sgarbas. Many people thing Quantum Computing is a hard and intimidating subject. This book is an attempt to change their minds and enlighten them.

9. **The Art of Computer Programming: Volume 4A** by Donald E. Knuth. Review by John D. Rogers. Say you want to list out all of the elements of $\{0,1\}^n$ quickly. How would you do it? How would you measure quickness? Say you wanted to list them out so that every strings differs from the prior one in only one bit (a Gray Code). Then how would you do it? This book is about how to quickly list out elements of a set. Given the author it has lots of historical context and mathematics of interest. Here is hoping we will one day see Volume 4B.

10. **Boolean Function Complexity: Advances and Frontiers** by Stasys Jukna. Review by William Gasarch. Given a Boolean function (e.g., Parity) and a model of computation (e.g., constant depth circuits) can you compute that function in that model? If so, what size does it take. For example, as is well known, constant depth circuits for parity require an exponential number of gates. This book goes through an enormous number of models and for each one gives upper and lower bounds on several Boolean functions. The book includes both well known and very little known (but interesting) results.

BOOKS I NEED REVIEWED FOR SIGACT NEWS COLUMN

Logic and Computability

1. *Computability and Complexity Theory (2nd Edition)* by Homer and Selman.
2. *Proof Analysis: A Contribution to Hilbert's Last Problem* by Negri and Von Plato.
3. *Programming with Higher Order Logic* by Miller and Nadathur.
4. *Graph Structure and Monadic Second-Order Logic: A language -theoretic approach* by Courcelle and Engelfriet.
5. *Software Abstractions: Logic, Language, and Analysis* by Daniel Jackson.

Algorithms, Cryptography, and Coding

1. *Integrated Methods for Optimization (second edition)* by John Hooke
2. *Algebraic Shift Register Sequences* by Goresky and Klapper.
3. *Information Theory and Best Practices in the IT industry* by Sanjay Mohapatra.

Misc

1. *A Wealth of Numbers: An anthology of 500 years of popular math writings* Edited by Benjamin Wardhaugh.

Joint Review [2] **of**
How to Solve It: A New Aspect of Mathematical Method
George Polya
Princeton University Press, 2004
253 pages, Softcover
and
Street-Fighting Mathematics
Sanjoy Mahajan
The MIT Press, 2010
134 pages, Softcover

Review by
Mohsen Vakilian

1 Introduction

Both books teach you some techniques to solve mathematical problems. The book by Polya is a classic and more general book whose techniques can be applied when solving any mathematical problem. But, the book by Mahajan presents some specific tricks to get you to the answer quickly.

2 Review of How to Solve It: A New Aspect of Mathematical Method

This book was printed in 1945 for the first time. It is a seminal book on problem solving techniques by a famous mathematician, George Polya.

Polya tries to teach teachers how to teach problem solving to students. He breaks down the steps to solve a problem into four parts: *understanding the problem*, *devising a plan*, *carrying out the plan* and *looking back*. Then, he breaks down each step even further by listing questions that the teacher could ask the student to guide him/her in the process of solving the problem. As Polya describes it, these questions have two common characteristics: *common sense* and *generality*. Polya's list of questions help the teacher and student build a constructive dialog.

In part one, Polya briefly describes the four steps of solving a problem and gives a simple example to present his method. In part two, he gives some suggestions on how to start solving a problem and how to hunt for helpful ideas. Part three is the main part of the book that shows how Polya's list of questions help students solve problem. This part starts by discussing several heuristics such as *analogy* and *auxiliary elements*. It then continues by giving suggestions on how to check the result and derive it differently. Polya distinguishes devising a plan from carrying it out by saying that we may use provisional and merely plausible arguments when devising the plan. But, the result of carrying out the plan should be supported by rigorous arguments. Polya categorizes problems as "problems to find" and "problems to prove". Then, he continues by giving some heuristics specific to each kind of problems. For example, for "problems to find" he suggests

the problem solver to think about questions such as: "Could you derive something useful from the data?" and "Could you restate the problem?".

The book is full of techniques and heuristics such as *generalization*, *induction*, *signs of progress*, *specialization*, *subconscious work*, *symmetry*, *test by dimension*, *variation of the problem* and *working backwards*. But, this book is not just a catalog of mathematical techniques for problem solving. Rather, it also considers the psychology of problem solving. There are lots of examples of dialogs between the instructor and student, where the instructor asks a sequence of questions from the student that lead the student to come up with the solution by himself/herself. Polya explicitly says that problem solving is not a purely "intellectual affair"; determination and emotions play an important role.

3 Review of Street-Fighting Mathematics

This book presents some informal techniques to solve mathematical problems. These techniques usually do not lead to rigorous proofs. But, they provide alternative ways to check and generate answers that are faster and are more intuitive than rigorous arguments. The following is a brief summary of the techniques presented in each chapter. These techniques might seem familiar to you. But, you will be amazed to see how far these techniques can go in solving nontrivial examples of the book.

The first chapter presents *dimensional analysis*. Dimensional analysis is based on the fact that only entities with the same dimensions are comparable. This chapter uses dimensional analysis to solve some nontrivial problems such as finding $\int \frac{dx}{x^2+a^2}$.

The second chapter describes how one could guess the form of the solution by computing it in cases that are easy.

The third chapter discusses techniques to approximate various quantities. This chapter starts by calculating the number of babies in the US without using the detailed data published by the US Census Bureau. The rest of the chapter gives example of how to estimate integrals, factorials and derivatives and wraps up by doing a more practical example, i.e. predicting the period of a pendulum.

Chapter four presents pictorial proofs as a companion to symbolic proofs. As an example, it shows why the sum of the first n odd numbers is n^2 by putting together the corresponding L-shaped puzzle pieces into a square.

Chapter five is about successive approximation, i.e. computing expressions by iteratively taking out the big parts and then considering the correction parts. For example, $(3 + 0.15)(7 + 0.21)$ has a big part of 3×7 and a correction part of roughly 5% (from 0.15 relative to 3) plus 3% (from 0.21 relative to 7), making a total correction of 8% of 21, for a total of 22.68 (the exact product is just over 22.71.)

The sixth chapter introduces *analogy*, the technique that the author describes as the final street-fighting tool of reasoning. Analogy is about finding a similar problem that is easier to solve. The author motivates this technique by a spatial trigonometry example in three dimensions, and solves the problem using an analogous problem in two dimensions.

4 Comparison

If you teach a mathematical subject you must read How to Solve It. Also, if you are a student who wants to improve his/her problem solving skills you will find How to Solve It incredibly useful. It gives you a list of questions that you can ask yourself while solving a problem on your own.

On the other hand, if you enjoy finding answers quickly without going through the rigorous proofs, you will find Street-Fighting Mathematics an interesting book. This book will help you strengthen your intuitive insight in solving problems.

The goal of How to Solve It is to articulate a general process for solving problems. But, Street-Fighting Mathematics focuses on six tricks that you could use to solve certain problems quickly. Some of the techniques in Street-Fighting Mathematics are also covered in How to Solve It. But, Street-Fighting tends to give more examples and show that its simple techniques could be applied to nontrivial problems. In general, techniques in Street-Fighting Mathematics could be used when devising a plan to solve a problem based on How to Solve It.

Review of[3]
Grammatical inference: learning automata and grammars
by Colin de la Higuera
Cambridge University Press, 2010. ISBN: 9780521763165
485 pages. Hardcover, $85

Review by
Matthias Gallé, `matthias.galle@xrce.xerox.com`
6 Chemin Maupertuis, 38240 Meylan, France

1 Introduction

Grammatical Inference studies how to learn automata and grammars from a set of observations. It differentiates from other similar tasks from Machine Learning by the underlying assumption that the target that generated the observations is some kind of formal grammar, and by its emphasis to identify exactly the original gold grammar, instead of just finding one that explains well the observations. It is a research field by its own, with a flagship conference (the International Colloquium on Grammatical Inference) held since 1994 but until now no book covering the field was available. Colin de la Higuera fills this gap with this book which will most probably become *the* book of the field.

2 Summary

The book is divided into three parts. Besides this, there are two introductory chapters which motivate the field by some toy examples and give an idea of the challenges faced; and a concluding last chapter highlighting some points which were mentioned during the book.

2.1 Part I

The first part includes four chapters that give separately the basic bricks upon which the rest of the book is based. It overviews stringology/text algorithms (Ch. 3), formal grammars (Ch. 4), probabilistic distributions over strings defined by formal grammars (Ch. 5) and associated combinatoric problems (Ch. 5). Chapter 3 covers string order and distances and kernels. Chapter 4 gives the classical definitions of (non-)deterministic finite-state machine, context-free grammars and some other classes. Chapter 5 treats probabilistic version of these and define several notions of distances between the resulting distributions. Chapter 6 gives a short overview of VC-dimensions and defines precisely the typical search space for deterministic finite state automata.

2.2 Part II

This second part deals with different paradigms of learnability. This is, what it means for an algorithm to learn a grammar and which are the available mechanism it has to do so. It also covers four chapters. The long Chapter 7 deals with the definition of a classical paradigm called

Identification in the Limit. A good part of the chapter deals with the different ways to count in order to define polynomial time learning (count number of mind changes, number of examples, etc). The different settings covered are learning from text (positive examples only), from an informant (positive and negative examples), from queries (active learning) and learning with noisy data. Chapter 8 presents a series of results concerning learning in the limit from text, and Chapter 9 does the same for learning with queries. Finally, Chapter 10 presents PAC-learning (and variants) in grammatical inference land.

2.3 Part III

This last part reviews different techniques and presents algorithms to learn some classes of languages under the settings defined previously. Chapter 11 analyzes existing algorithms for learning in the limit different languages classes from text only. Chapter 12 reviews two classical algorithm for learning deterministic finite state automata from an informant. Chapter 13 concerns learning with queries by analyzing the LSTAR algorithm. Chapter 14 diverges a bit from the schema until now by presenting different heuristics used in the field, that, while not ensuring necessarily any theoretical condition of learnability, work good in practice. Chapter 15 concerns the difficult task of learning context-free languages and grammars, with different approaches. Chapter 16 tackles the problem of learning probabilistic finite-state automaton (both the structure and the probabilities) while Chapter 17 presents well known algorithms to learn the probabilities when the structure is known (the Baum-Welch and Inside-Outside algorithm). Finally, Chapter 18 covers transducer and learnability issues associated with them.

3 Opinion

In my opinion this book will be fundamental to the field for two reasons: Firstly: it sets a unified notation and the author has been very careful in defining his concepts in such a way that they can be reused by several of the algorithms. In a small field like Grammatical Inference this will without doubt ease the integration of new contributions. Secondly, the book serves as a reference book for anyone interested in the field. This applies particularly for Part III, where the most classical algorithms are presented in a unified way, together with their main properties. It is understandable that many variations of these algorithms, or other secondary ones, did not find their place in the book (although they may be found in the references). Similarly, applications of these algorithms are almost not mentioned. On one hand they are out-of-scope of the book, but traditionally the grammatical inference community is mainly interested in the principles behind the definitions and algorithms and successful applications find their place in the corresponding community.

Probably the person that will take the most advantage of the book is the graduate student working in Grammatical Inference; or the researcher that found an interesting result and would like to put it into perspective, or thinks he has a problem which can be solved by some of these techniques. It should be noted that the author divided clearly the definitions of the tools (Part I), definitions of learnability and general results (Part II) and presentation of algorithms (Part III), so that even the occasional reader will probably have to jump forward and back through the book. Should you read this book? If you are interested in the field or you think you could do a contribution, absolutely. If not, you may want to have a look at Part II, some chapters of Part III and the conclusions to add to your general knowledge.

Some of the most valuable parts of the book are the last two sections of each chapter, namely "Alternative lines of research" and "Open problems and possible new lines of research". Anyone interested in the field will find here plenty of inspiration to tackle open problems. Each chapter concludes with a list of exercise of very different degree of difficulty. They are probably not enough for a course by themselves, but they help to grasp the concepts.

Review of[4]
Logical Foundation of Proof Complexity
by Stephen Cook and Phuong Nguyen
ASL PUBLISHER, 2010
479 pages, HARDCOVER, $70.00

Review by
Arthur MILCHIOR (arthur.milchior@ens.fr)

1 Introduction

First we give an informal example. Let $A \in \mathsf{P}$ (polynomial time). Is there a formula ϕ (in some language that we have not specified yet) with one free variable such that

$$A = \{x \mid \phi(x)\}.$$

If $x \in A$ we should be able to produce a proof of $\phi(x)$. If $x \notin A$ we should be able to produce a proof of $\neg\phi(x)$. How does the size of the proof vary with $|x|$?

We now describe a theorem about P that is proven in this book. The logic V^1 (informally) has the axioms of Peano Arithmetic and a few axioms about strings of bits, plus the comprehension scheme over existential formulae.

Theorem: For any set $A \in P$ there is a first order formula $\phi(x)$ such that the following hold.

1. ϕ begins with an existential quantification of a string of bits.
2. After that first quantifier there are bounded quantifiers over the integers where the bound is a polynomial in the length of the input.
3. For all $x \in A$ the statement $\phi(x)$ can be proven in V^1 in a polynomial (in $|x|$) number of steps.
4. For all $x \notin A$ the statement $\neg\phi(x)$ can be proven in V^1 in a polynomial (in $|x|$) number of steps.

In general, there are two important things to consider when studying expressivity— (1) what is the set of formulae one wants to prove, and (2) what is the set of axioms you are allowed to use. For the sake of simplicity in this review I will write "a logic" instead of "a choice of set of formulae and axioms".

The proof systems can use either an induction scheme, a minimization scheme, or a comprehension scheme, or any combination. An induction scheme is the usual one in mathematics: if $P(0)$ and $P(i) \Rightarrow P(i+1)$ then $(\forall i)[P(i)]$. A minimization scheme says that if there exist an integer i such that $\phi(i)$ is true, then there is minimal such i. A comprehension scheme means that, for each formula $\phi(x)$ and integer y, there is a string B of y bits such that the ith bit of B is 1 if and only if $\phi(i)$ is true. By restricting the formulae that can be used in a scheme, for example, by bounding the number of alternation of quantifiers one can obtain language which is more or less expressive.

Proofs can also be propositional. In this case there are no numbers and string of bits, but only boolean variables. Some proof systems can accept an ad hoc axiom to state that some problem, complete for some complexity class, has a computable answer.

2 Summary

Chapter I gives a quick introduction to the domain, introducing most of the logics without defining them. It gives a good idea of what type of results will be proved, but since the vocabulary used is not yet known, during the first reading of this chapter it is not easy to understand what this really means.

Chapter II is an introduction to logic with classic definitions and theorems. Here we find the definition of the syntax and semantic of the propositional and predicate calculus, correctness and completeness results, the "equality" definition in a first order logic, the existence of a prenex normal form and the Herbrand theorem. This chapter clarifies the notation and gives examples of theorems.

Chapter III formally defines several notions that are used in the first part of the book:

1. Peano Arithmetic
2. A polynomial-bounded theory, which means functions bounded by a polynomial such that the formulae to define them use polynomially bounded quantification.
3. A conservative extension, which means that if you can define a function f in a logic, then you can add f in the vocabulary and add its defining axiom to the theory without changing the expressivity.

This chapter also gives the first example of relation between a logic and a complexity class. They prove that the linear time hierarchy is equal to $I\Delta_0$, the set of functions that one can define with Peano's axioms plus induction over bounded formulae.

Up to Chapter III, all quantifications are over integers. Chapter IV introduces quantification over strings of bits. Often a formula uses both kinds of quantification. Such formulae are called 2-sorted. Two functions exist over strings, the size of the string and the value of one bit. They explain the reason for the choice of using two sorts, mostly that the length is logarithmic in the number represented by the string, and so polynomial bounds can be defined to be applied only to integers, and not to the string of bits seen as a number, which would give exponentially larger bounds. Then they extend definitions and theorems from single-sorted logic to the two sorted one.

Chapter V and VI do not introduce new fundamental notions, but build on the definitions of previous chapters to prove (1) the equivalence between the logic V^0 and the complexity classes AC^0 which is the smallest interesting class in complexity and (2) the equivalence between V^1 and P. They also prove the equivalence between a few different definitions of the same logic, and introduce some new notions, like the "witness", the fact that if a formula begins with universal quantifiers, and then existential quantifiers, one can replace the existentially quantified variable by a function depending on the universally quantified one and the "replacement axiom scheme".

Chapter VII studies propositional logic, where all the variables are boolean. The interest is now on families of formulae, the axioms needed to prove them, and the size of the proofs. Predicate logic's results are transformed into propositional logic's results by translating the formulae and the proofs from one formalism to the other.

Chapter VIII and IX define a number of important logics and explain how one logic can efficiently simulate another logic, and how they relate to complexity classes. Chapter VIII is about logics related to complexity classes that contain P. Chapter IX is about logics related with smaller complexity classes. Both chapter state some open problems about how new result in complexity or in logic would affect the other field. For example in chapter VIII, there are results about logics without strings and Horn logic.

Chapter IX gives more details of what families of formula can and cannot be proved in some logic. The most used example is the "pigeon hole principle" and some variation of it. In particular, this chapter discusses the question of finding the smallest axiom set possible to prove some family of theorems. This chapter represents almost one fourth of the entire book; there is an impressive amount of theory and result.

Finally Chapter X comes back to quantified propositional logic; in chapter VII first order logic was translated to propositional Logic. By contrast this time propositional logic is coded into first-order logic using the "reflection principle". The truth of a family of propositional formulae and the correctness of their proof is defined in some first order logic. This second direction increases the importance of the relation between those two kinds of logic. In particular the translations are as efficient as possible while respecting soundness.

3 Opinion

This books seems intended for graduate students and researchers in the computational complexity or logic. It requires a prior familiarity with basic complexity results and logic (both proof theory and model theory). Even though Chapter II and Appendix A give an introduction to logic and complexity theory, it is not enough background for the reader to understand the concepts used throughout the entire book.

The main issue with this book is the quantity of notations used, and that most of it is given without any explanation about the choices. Many articles uses different notations— this is a standard problem in the field of logic— and many authors give different names for the same theory. Hence it is great to have everything unified in one book. But the cost is that often a theory's name is only one or two letters long, sometime with a number, and no explanation is given for the name. For example the theory V^i or the System G have names that are hard to remember since its hard to remember what the letters stand for. Also consider that Σ_i^b and Σ_i^B, where i is an integer, are two different theories whose definitions look alike.

There is an index at the end of the book, but it does not always help as much as one could hope. For example to find "pBNR" you need to search under "B", and to find "Σ_i^B" you need to recall that it is a formula classes. Note that "Σ_i^b" and "$\Sigma_j^B(V^i)$" are not near each other.

Most of the proofs are technical but there is no really subtle idea that would be hard to grasp. In each section, the first proofs are done with enough detail to correctly follow them when we remember the definitions; then the next proof of the same type are shorter or left as exercises, which avoid useless repetition.

For those two reasons, one often has to trust the authors for the correctness of the demonstrations; checking in details every theorem would take a lot of time or an excellent memory, since

often, many theorems are not very different, but they have tedious details one has to check. On the other hand it is an excellent book for someone who wants to learn what equivalences between logic and small complexity classes are known.

Review of[5]
Exact Exponential Algorithms
by Fedor V. Fomin and Dieter Kratsch
Springer, 2010
203 pages, hardcover, $60.00

Review by
Michael Lampis mlampis@kth.se
KTH Royal Institute of Technology, Stockholm, Sweden

1 Introduction

It's a well known fact of computational complexity theory that the vast majority of interesting algorithmic problems are, unfortunately, intractable. In other words, thanks to the theory of NP-completeness, we know that for literally thousands of optimization problems it is impossible to obtain a polynomial-time algorithm that produces an exact solution (unless P=NP). Still, all these problems need to be solved, so what are we to do? The approach that has probably been most favored in the theoretical computer science community is to "approximate", that is, try to design an efficient algorithm that gives a solution that is as good as possible. However, in many cases this is not appropriate, either because any sub-optimal solution is unacceptable or because even obtaining a decent approximate solution for our problem is hard. This is the motivation of this book, which deals with algorithms that always produce optimal solutions - exact algorithms.

Of course, assuming standard complexity assumptions, an algorithm that always produces optimal solutions for an NP-hard problem requires super-polynomial (usually exponential) time. In most cases obtaining a brute-force exact algorithm is easy: e.g. it's trivial to solve n-variable SAT in time (about) 2^n and TSP on n cities in time $n!$. The important point though is that in most cases this is far from the best possible (for example TSP is solvable in $O(n^2 2^n)$). How can one achieve such improvements? This is exactly the topic of this book.

2 Summary

This is a book that presents algorithmic techniques which can be used to obtain improved exponential-time algorithms for various algorithmic problems. To put it simply, the typical problem this book tries to attack is the following: for a specific NP-hard problem (often on graphs) we already know a c^n algorithm. Can we get an exponential algorithm with a smaller base? Similarly, can we improve $n!$ algorithms to c^n algorithms? The book is written mainly for graduate students or researchers interested in an introduction to this research area, which after many years of neglect has seen a significant amount of growth in the recent past.

The book is divided in chapters that explain different techniques, starting off from the simpler ones (branching algorithms, dynamic programming) and moving on to more complex tools such as subset convolution. In every chapter a technique is illustrated with several example applications to concrete problems.

More specifically the topics covered in each chapter are as follows:

[5]

- Chapter 1 is a general, gentle introduction to the subject matter. It introduces notation that will be used throughout the book, such as the O^* notation, which suppresses polynomial factors, and is thus appropriate to use in the study of exponential algorithms. There is also a discussion of a topic that is often glossed over in other fields of algorithms research but is especially important here: how do we measure the size of the input? For an exponential-time algorithm it makes a huge difference whether we consider n to be, say, the number of vertices or the number of edges of the input graph. Though there is no uniform answer to this question, the approach the authors follow is to use the measure of the size of the input that is most "natural", e.g. the number of vertices for graph problems or the number of variables for satisfiability problems. Finally, two classical results are presented here: the Held-Karp algorithm that solves TSP in $O^*(2^n)$ time and a branching algorithm for independent set running in $O^*(3^{n/3})$.

- Chapter 2 talks about branching algorithms. This is the most basic and easy to understand technique in the book. The basic idea is that the algorithm makes a simple choice at each step (e.g. whether to select a specific vertex as part of an independent set or not) and based on it "branches" into several instances of the problem which are solved recursively. Rather than the algorithms themselves, the devil here is in the details of the analysis, which involves solving recurrence equations. The topic is treated in detail here and two full examples are given for SAT and Independent Set.

- Chapter 3 covers dynamic programming, a technique which is often well-known for its application to polynomial-time algorithms. The authors discuss (with examples) how it can be used to obtain c^n algorithm for permutation problems, where the brute-force approach would take $n!$. Another interesting result here is a 2^n algorithm for Set Cover, where n is the number of elements in the universe (it's trivial to obtain a 2^m algorithm where m is the number of sets). An important point discussed here, which will be returned to later, is that dynamic programming algorithms often need exponential space in addition to exponential time, something which may cause important complications in practice.

- Chapter 4 discusses Inclusion-Exclusion. This is the first technique discussed that is both non-trivial and unique to exponential-time algorithms. As the name suggests, it is a technique based on the well-known inclusion-exclusion principle from combinatorics, which is a formula to calculate the size of the union of a number of sets. Inclusion-exclusion can be used to solve a number of hard (#P-complete) counting problems, by calculating the answer as the sum of the (usually 2^n) terms of an inclusion-exclusion formula. As a consequence of solving the counting problem, one also obtains a solution for the decision version, and several examples are given in this chapter, including the breakthrough results of Björklund and Husfeldt, and Koivisto for calculating the chromatic number in $O^*(2^n)$ time.

- Chapter 5 discusses treewidth, a measure that quantifies how close a graph is to a tree. Treewidth has played a major role in the area of parameterized complexity, a field that is a close cousin to the area of exponential algorithms, and the authors discuss here how this tool can be used to obtain improved exponential-time algorithms using dynamic programming on tree decompositions. They also discuss the problem of actually computing the treewidth of a graph, a problem that is itself NP-hard.

- Chapter 6 returns to the basic idea of branching algorithms but refines the analysis even further. Normally, we measure the amount of progress that an algorithm makes by the size of the instance (e.g. the number of vertices of a graph), and this is the variable of the recurrence relations we solve to get a bound on the running time. The analysis can be refined by using a more sophisticated measure (e.g. assign weights to the vertices according to their degrees), an approach called Measure & Conquer. Surprisingly, this more sophisticated analysis can yield a better running time without actually changing the branching algorithm! Things become quite involved pretty quickly here, and there's endless room for the definition of different measures. It's telling that there's also a section on lower bounds: it deals with the problem of finding families of hard instances that give lower bounds on the running times of specific algorithms, thus finding limits on how far the measure & conquer approach can improve the running time of an algorithm.

- Chapter 7 discusses the operation of Subset Convolution. It introduces tools such as the zeta and Möbius transform to speed up the trivial 3^n algorithm for calculating the convolution of two functions of subsets of an n-element set to 2^n. This probably doesn't sound all too exciting, but it turns out that it's actually a pretty big deal, because a lot of standard problems can be expressed as a convolution of two functions (e.g. calculating the chromatic polynomial). Thus, this is a nice framework to get 2^n algorithm for these problems. The authors also make the interesting point that a lot of the ideas here are very similar to those used in Fast Fourier Transform calculations.

- Chapter 8, which is pretty short, deals with local search algorithm, mainly applied to the satisfiability problem. The idea here is again simple (start with a random assignment and as long as it's not satisfying randomly pick an unsatisfied clause and fix it) but the analysis is more complicated. The authors also spend a little time discussing how to get a non-randomized version.

- Chapter 9, which is also short, is actually one of my favorites because it takes results that everyone knows and uses them in an unexpected way to produce something new. It discusses a technique called Split & List, which basically applies non-trivial polynomial-time algorithms for simple problems to obtain non-trivial exponential-time algorithms. More specifically, it makes use of the fact that sorting can be done in time $O(n \log n)$ (rather than the trivial $O(n^2)$) and matrix multiplication in time $O(n^\omega)$ with $\omega < 2.376$ (rather than $O(n^3)$) to obtain better algorithms for Subset Sum and Max Cut.

- Chapter 10 goes back to the problem of space usage in dynamic programming algorithms. It discusses ways to make the (exponential-size) dynamic programming table smaller, at the expense of the running time. The theme here is that we try to come up with algorithms that work in time $T(n)$ and space $S(n)$ so that $T \cdot S$ is as small as possible, or we can get good trade-offs. Another extension to this theme is a discussion of how branching algorithms (which usually only need polynomial space) can be made faster by using memorization, that is, storing the answer to (an exponential number of) small instances.

- Chapter 11 is a collection of three mostly unrelated topics. First, there is a discussion of the Bandwidth problem. Second, there is a short discussion of a technique called Branch & Recharge which extends the ideas of Chapter 6. And finally, there is a discussion on lower

bounds and the Exponential-Time Hypothesis (ETH), which states that there is no $2^{o(n)}$ algorithm for 3-SAT. It is explained how this is equivalent to the hypothesis that there is no $2^{o(m)}$ algorithm for 3-SAT (where m is the number of clauses), though the technical details of this major result by Impagliazzo, Paturi and Zane are not given here. From this, the authors sketch how it is possible to rule out the existence of sub-exponential time algorithms for a number of problems, assuming the ETH.

- Chapter 12 is a collection of open problems. It contains both broad questions (e.g. how can we show lower bounds on the base of the best exponential algorithm for a problem) and a bunch of concrete problems (e.g. can we solve Subgraph Isomorphism in $2^{O(n)}$). Many pointers are given to the literature, even to papers that were published only a few months before this book.

3 Opinion

Many theoretical computer scientists have been raised with the mantra "Polynomial good, Exponential bad". As a result, most of our community's research has focused on trying to design good (i.e. polynomial) algorithms (or trying to prove that this is impossible). What is the point of trying to improve a bad 2^n algorithm to, say, 1.8^n?

Of course this point of view, where the land of exponential time is a barren wasteland filled with worthless algorithms is far from the truth and it's a fact of life that exponential-time algorithms will have to be used often in practice. So, it's high time that we stopped looking down upon improvements on the constants in the bases of exponentials, as they can have a big effect in practice. Rather, we need to develop a theory of exponential algorithms that will allows us to make them as efficient as possible. This is the thesis of this book, and in my opinion it is quite correct. Furthermore, this book fills up a void in the literature by being (to the best of my knowledge) the first book devoted entirely to the study of this subject from the theoretical point of view.

I think the book does a good job of introducing the subject and the various techniques that the authors want to cover. It's written clearly and concisely and many examples are given. A CS graduate student should be able to pick up this book and be up to speed with the most important facts in the theory of exponential-time algorithms in a few weeks. Another major plus is that the book is very up-to-date, even mentioning results that were published mere months before it was printed. This makes the open problems chapter a very interesting read.

The book is not too large (around 200 pages) but I don't blame the authors for not including more material. I believe they cover all the major techniques and putting more results in would either make things repetitive or very technical (though more exercises would help). Some sections do feel like they need a lot more to be said, but I think the problem here is that the results have not yet been discovered, rather than that the authors left them out! (I'm mostly thinking of the section on the ETH and lower bounds here). I do look forward to reading the second edition of this book in 5-10 years, when the area will have matured even more!

This is an interesting and fun book to read. The authors did a good job of dividing the material into chapters and writing up both good motivating comments and readable proofs. Misprints and typos are few and far between. The only thing that kind of spoils the fun is that much of the work is impossible to follow using just pencil and paper: often the authors will derive some funny constant as the solution to an equation that was (approximately) solved by a computer. Unless

you are willing to run Mathematica while reading the book this will require you to "just believe" many of the results presented. Of course, this is just a characteristic of the research presented and I doubt anyone could have done a better job of writing up these results than the authors did here.

I'm glad this book was written, I think it was very much needed and that the authors managed to beautifully present a self-contained overview of an important field. I'd recommend this book to anyone interested in the topic, either to do research or just out of general curiosity.

Review of[6] of
Bioinspired Computation in Combinatorial Optimization
by Frank Neumann and Carsten Witt
Springer, 2010, $85.00
216 pages, Hardcover

Review by
Sven Herrmann s.herrmann@uea.ac.uk
School of Computing Sciences, University of East Anglia, Norwich, UK

1 Introduction

The book discussed in this review deals with algorithms inspired by mechanisms in biology and their use in combinatorial optimization. Bio-inspired algorithms are (usually stochastic) search algorithms that are inspired by processes in nature. In particular, this includes evolutionary algorithms, an approach that has become quite popular in the last 40 years. These algorithms try to solve problems by evolving through the search space until satisfying results are obtained. Another bio-inspired method is called ant-colony optimization. In this method, solutions are obtained by random walks on a so-called construction graph, modeling how ant colonies distribute information.

In combinatorial optimization problems, the aim is to minimize (or maximize) a (usually real-valued) function over a discrete and usually finite search space. In general, this space can be fairly arbitrary, but it usually consists of certain combinatorial objects, for example subsets of a set of a certain cardinality or paths in graphs. A lot of these problems can be modeled by weighted graphs, that is, graphs were each edge is equipped with a real-valued weight. Famous examples include the computation of minimum spanning trees or shortest paths in such a graph.

Given an algorithm to solve such a problem, one is usually interested in the time the algorithm needs (measured in the size of the input) to produce a solution. In the case of a stochastic or random algorithm, one can often not expect that the algorithm is guaranteed to produce a result at all in finite time, but one is mainly interested in the expected time it takes to arrive at the solution. The major part of the book is devoted to such runtime analysis.

2 Summary

The book is divided in three parts. In the first part, the basic concepts both from combinatorial optimization and stochastic search algorithms and their analysis are introduced. In the two subsequent parts, the general algorithms are applied to concrete combinatorial optimization problems with a single and multiple objectives, respectively.

2.1 Part I Basics

The first part contains a short introduction together with three chapters on basics in combinatorial optimization and stochastic search algorithms.

Chapter 2 Combinatorial Optimization and Computational Complexity

The book starts with a short discussion and definition of basic concepts of combinatorial optimization and the main objects: finite graphs. Then basic notions of complexity theory are discussed briefly, and the problem of data representation is addressed. This also includes a discussion of approximation algorithms and randomized algorithms. The chapter closes with a brief section about multi-objective optimization including the introduction of important concepts such as the Pareto front.

Chapter 3 Stochastic Search Algorithms

This chapter introduces the concepts of the several stochastic search algorithms mainly of the bio-inspired kind. In particular, evolutionary algorithms and their major approaches, namely evolution strategies, genetic algorithms, evolutionary programming and genetic programming, are discussed. Furthermore ant colony optimization and other (non bio-inspired) general stochastic search algorithms are briefly presented. It is pointed out that these algorithms can be easily adapted to almost any combinatorial optimization problem, although this comes with the drawback of not so rigorously analyzed runtime and/or approximation quality.

Chapter 4 Analyzing Stochastic Search Algorithms

Here, basic evolutionary algorithms are formally defined. These will be later adapted to concrete problems and formally analyzed. The algorithms are RLS_b^1, $\mathrm{RLS}_b^{1,2}$ (randomized local search) and $(1+1)\mathrm{EA}_b$ (a basic evolutionary algorithm) for single-objective optimization, and SEMO (simple evolutionary multi-objective optimizer) and GSEMO (global SEMO) for multi-objective optimization. After that, tools for analyzing the expected optimization time of these algorithms are given and some general theorems for the runtimes of these algorithms are presented.

2.2 Part II Single-objective Optimization

The second part contains five chapters analyzing the previously-defined algorithms for five concrete combinatorial optimization problems

Chapter 5 Minimum Spanning Trees

Given a graph $G = (V, E)$ with vertex set V and edge set E and a weight function $w : E \to \mathbb{R}^{>0}$, the aim of the minimum spanning tree problem is to find a subset T of E forming a tree such that $w(T) = \sum_{e \in E} w(e)$ is minimal among all such trees. There are classical (deterministic) algorithms by Kruskal and Prim to solve this problem efficiently. After an introduction of these concepts, it is explained how the different stochastical search algorithms introduced in Chapter 4 can be adopted to solve this problem. Furthermore, some speed-up techniques are discussed and explicit versions of ant-colony optimization adapted to this problem are given. The expected optimization times are discussed and it is noted that evolutionary algorithms are not competitive with the classical algorithms but that several important variants of the problems are difficult to solve and the stochastic search algorithms are useful in this case.

Chapter 6 Maximum Matchings

In the maximum matching problem, one searches a matching (that is, a subset $E' \subset E$ where no two elements of E' share a vertex) with maximal cardinality of a graph $G = (V, E)$ with maximal cardinality. This problem can be solved in polynomial time with the concept of so-called augmenting paths. This concept is discussed briefly, before the expected runtime of the randomized local search and basic evolutionary algorithm to find approximations to the problem are deduced. A deeper analysis then also presents expected runtimes for finding an exact solution. In the end, a certain graph is constructed for which the expected optimization time of these two stochastic algorithms is particularly bad.

Chapter 7 Makespan Scheduling

Given n jobs with positive processing times $p_1, \ldots, p_n$ and two identical machines to process them, the aim of the makespan scheduling problem is to distribute these jobs in such a way that the overall completion time is minimized. This problem is NP-hard in general and its decision variant is NP-complete. However, there are efficient approximation algorithms. Stochastical search algorithms for the problem are then introduced and analyzed. In the worst-case model, it is shown that in polynomial time the approximation ratio cannot be better than 4/3. By using suitable variants of the algorithms and multistart techniques, these results are enhanced drastically. In addition, two different average-case models are investigated, and it is shown that with growing problem sizes the evolutionary algorithms perform quite well.

Chapter 8 Shortest Paths

The shortest path problem for a graph $G = (V, E)$ together with a weight function $w : E \to \mathbb{R}_{>0}$ comes in two variants: SSSP (single-source shortest path) were the shortest paths from a given vertex $v \in V$ to all other vertices are to be computed, and APSP (all-pairs shortest paths), where for each pair $v, w \in V$ the shortest path between v and w should be obtained. These problems are classically solved by the algorithms of Dijkstra and Floyd-Warshall, respectively. For both, SSSP and APSP evolutionary algorithms are given and the expected optimization times analyzed. Additionally, ant-colony optimization is applied to the two problems, and a variant using interaction is given having better runtime bounds.

Chapter 9 Eulerian Cycles

A Eulerian cycle in a graph $G = (V, E)$ is a cycle that uses each $e \in E$ exactly once. It is easily seen that such a cycle exists if and only if all $v \in V$ are contained in an even number of edges; such graphs are called Eulerian. Given the knowledge that a given graph is Eulerian, the task is now to compute a Eulerian cycle. A simple deterministic algorithm of Hierholzer solves this problem in time linear in the number of vertices and edges. Although it is stated in advance that stochastic search algorithms are not expected to perform equally well, different evolutionary algorithms are developed and investigated, and it is shown how the expected runtime can be improved using adjacency list matchings.

2.3 Part III Multi-objective Optimization

In the third and last part of the book, multi-objective optimization via bio-inspired algorithms is discussed. This includes genuine multi-objective optimization problems, but also combinatorial problems that can also be solved with single-objective methods, where the multi-objective approach leads to improvements. The book closes with a short appendix on probability distributions, deviation inequalities and other useful formulæ.

Chapter 10 Multi-objective Minimum Spanning Trees

The multi-objective minimum spanning tree problem deals with finding spanning trees of a graph with respect to not one, but several weight functions on its edges. This problem is NP-hard even in the case of two weight functions. It is shown that the GSEMO algorithm can be adapted to this problem to obtain an efficient algorithm that computes a spanning tree for each vector of a strongly convex Pareto front.

Chapter 11 Minimum Spanning Trees Made Easier

In this chapter, the standard (single-objective) minimum spanning tree problem is analyzed by multi-objective methods. It is shown that for randomly chosen dense graphs this approach is more efficient than the direct one, whereas for sparse connected graphs the single-objective variant of the algorithm performs better. In addition to these theoretical asymptotic results, the authors also performed experiments to justify that the use of the multi-objective methods for this single-objective problem makes sense in practice.

Chapter 12 Covering Problems

In the vertex-cover problem, the task is to find a minimum set $V' \subset V$ of vertices of a graph $G = (V, E)$ such that each edge contains at least one vertex from V'. This is a classical NP-hard problem. It is shown that it cannot be solved efficiently with evolutionary algorithms when considered as a single-objective optimization problem. In particular, it may happen that the algorithms get stuck at very bad solutions. However, the multi-objective algorithm GSEMO obtains good approximation ratios and they can even be enhanced by using a variant of this algorithm with an asymmetric mutation operator. It is also shown that for more general set-cover problems multi-objective algorithms likewise perform better than single-objective ones.

Chapter 13 Cutting Problems

The last problem considered in the book is the problem of finding a minimum cut in a given directed graph. Again, it is demonstrated that the single-objective algorithms RLS_b^1 and $(1 + 1)\mathrm{EA}_b$ have exponential expected runtime, whereas GSEMO can solve the problem efficiently and a variant, DEMO (Diversity Evolutionary Multi-Objective Optimizer), solves the problem in expected polynomial time. This is even true for the multicut problem, which deals with several sources and targets.

3 Opinion

The book is suitable for a course on theoretical aspects of bio-inspired computing. It contains the needed basics and accessible proofs for almost all results it contains. Furthermore, it might serve as a useful resource for researches interested to apply bio-inspired algorithms, but also for those researches that already use bio-inspired algorithms and need tools for analyzing them.

The book is generally well-written and very well readable also for non-experts of the field. Additionally, (most of) the explanations and proofs are sufficiently detailed and self-contained enough to make the book well usable as a textbook. It is very nice to have a good summary about know results for the theoretical background of bio-inspired computations, as those algorithms are widely applied but there seems to lack an overview on their asymptotic behavior. The book covers this very well and it does not lack stating when bio-inspired methods are inferior to conventional algorithms. Particularly interesting is the discussion of multi-objective approaches and the examples of problems that naturally come with a single objective, but where those algorithms fail and multi-objective approaches can be used successfully.

Review of[7]
Triangulations: Structure for Algorithms and Applications
by Jesús A. De Lorea, Jörg Rambau, and Francisco Santos
Springer, 2010,$76.00
535 pages, Hardcover

Review by
Michael Murphy piggymurph@gmail.com
Venice, CA

1 Introduction

Triangulations are mathematical objects used daily for piecewise-linear and higher-order approximation, with applications in computer graphics, tomography, and physical simulations. With no geometry attached, triangulations are often called "abstract simplicial complexes" and are fundamental objects in algebraic topology. Moreover, there is growing branch of physics concerned with quantum gravitation that employs "Causal Dynamical Triangulations" for its calculations. The study of these important structures gives rise to many interesting geometric, combinatorial, and algorithmic questions, which are treated nicely in this book.

2 Summary

The authors' main focus is on triangulations of a fixed point set living in some Euclidean space. Among the many triangulations of a point set, the most famous one is the Delaunay Triangulation. Many of the topics in this book can be easily grasped if one has a basic understanding of Delaunay Triangulations and two fundamental algorithms for constructing them: Lawson's flipping algorithm and the "lifting map." The Delaunay Triangulation of a set of points S in the plane is the set of all triangles with vertices in S whose circumcircles[8] are empty.

Lawson's flipping algorithm is one of the earliest and easiest methods to compute a Delaunay triangulation of a set of points in the plane. Start with any triangulation of S. If there exists a triangle t with vertices a, b, and c whose circumcircle contains a vertex d of a neighboring triangle $u = (b, c, d)$, then flip the shared edge (b, c) out, and replace it with the diagonal (a, d). This destroys t and u and creates new triangles (a, b, d) and (b, c, d), which are *Locally Delaunay.* We continue flipping until all triangles are Locally Delaunay, at which point we have arrived at the true Delaunay triangulation. One consequence of Lawson's algorithm is that two triangulations of a point set S can be related by a sequence of flips. Thus, the *flip graph* whose vertices are the triangulations of S, with an edge between two triangulations if they can be related by a flip, is always connected.

While the notions of triangulations and flipping can be generalized to higher dimensions, as is done carefully in the first few chapters, Lawson's algorithm does not readily generalize. Attempts to generalize Lawson's algorithm have been stymied by the inability to know whether the flip graphs

[7] ©2013, Michael Murphy
[8] The Circumcircle of a triangle is the circle that contains all three vertices.

of triangulations of point sets in higher dimensions are necessarily connected. The desire to answer these flip-graph connectivity questions motivates a fair amount of the material in this book.

When working with point sets in 3 and higher dimensions, one typically uses the "lifting map" method to construct Delaunay Triangulations, since it works in every dimension. This entails mapping every point in a given point set S in R^d onto a paraboloid in R^{d+1} taking the convex hull, and projecting the lower envelope of the hull back into R^d. Note that other triangulations of S can be obtained by lifting S to another convex surface, not just the paraboloid. If a triangulation of S can be obtained in such a way, it is called a *regular triangulation.*

With the machinery of higher-dimensional flipping and regular triangulations defined and illustrated in the first 4 chapters, the authors are ready to describe a stunning construction by Gelfand, Kapranov, and Zelevinski known as the "Secondary Polytope." For every triangulation T of a set of n points $S = x_1, \ldots, x_n$ in R^d, a point in R^n is calculated. The ith coordinate is computed by summing the total volume of all simplices incident upon x_i in T. The Secondary Polytope of S is the convex hull of this new point set. Amazingly, the vertices of the secondary polytopes are the regular triangulations of S. And the edges of this polytope connect regular triangulations that differ by a flip. Secondary Polytopes have many other beautiful properties that the authors clearly describe.

It is easy to see by using Secondary Polytopes that the flip graph of a point set's regular triangulations is connected. However, it is the case that "many" triangulations of a point set in R^d are not regular, as seen in Chapter 6. Chapter 7 is devoted to interesting examples concocted by the authors that show that the flip graph of point sets in 4 dimensions and higher can be disconnected. They also give examples of "flip-deficient" point sets that suggest that there might exist point sets in R^3 with disconnected flip graphs. However, flip-graph connectivity for triangulations of three-dimensional point sets remains an outstanding open question in Discrete and Computational Geometry.

Although flip-graph connectivity and regular triangulations are a major focus of the book, there are quite a few more topics that are covered. Chapter 8 focuses on algorithmic and complexity questions related to triangulations, including enumeration, NP-completeness of Minimum Weight Triangulation, and linear programming. Chapter 9 describes constructs and techniques further relating triangulations to convex polytopes, including hints at the methods used by one of the authors (Santos) to disprove the famous Hirsch conjecture.

3 Opinion

This book is not for everyone who might be interested in triangulations. Someone looking for algorithms to make unstructured three-dimensional meshes of complicated domains for finite-element computations will be disappointed, as will topologists interested in homology and homotopy calculations of large simplicial complexes. Physicists working on Monte Carlo Quantum Gravity simulations with Dynamical Triangulations should take careful note of the results concerning flip-graph connectivity, however, since they often incorrectly assume that flip graphs are connected. People interested in Computational Geometry, Linear Programming, and Convex Polytopes should find the many illustrations and nice exposition make the book a joy to read.

Review of[9] of
Flows in Networks
by L. R. Ford Jr. and D. R. Fulkerson
Princeton University Press, 2010, $30.00
208 pages, softcover

Review by
Yixin Cao yixin@sztaki.hu
Computer and Automation Research Institute, Hungarian Academy of Sciences

1 Introduction

Since their inception, the cut and flow problems, as well as their extensions and generalizations, have been at the heart of algorithmic research on (di)graphs for over 50 years. There is a rich literature on this topic that abounds in all major conference proceedings and journals of theoretical computer science. Their applications are ubiquitous and some of them are greatly important, e.g. the only mathematicians who have ever been awarded Nobel Prizes, on market equilibria in 1994 and on stable market allocations in 2012, are recognized for their work on applying linear program duality and bipartite matching in economics, both are essentially related to network flows and studied in the book under review.

"Flows in Networks" was written by the co-inventors of this theory exactly half a century ago. With a new foreword by Robert G. Bland and James B. Orlin, Princeton University Press resurrected it as a volume of its series "Princeton Landmarks in Mathematics and Physics ". This presents us a chance to trace back to the origin and elements of this powerful theory. A similar act was performed by AMS Chelsea Publishing, who republished in 2009 "Matching Theory" by László Lovász and Michael D. Plummer (originally appeared in 1986).

2 Summary

It might help to point out that in the new foreword Bland and Orlin have made a very detailed explanation on the material.

Chapter 1 founds the base of the whole book. The famous max-flow min-cut theorem, was established as soon as the required notations were defined. It is followed by a labeling based algorithm that computes maximal flows *efficiently*. The second part of this chapter extends and amplifies this theorem from several aspects, e.g. allowing multiple sources and sinks; and imposing capacities on nodes instead of arcs. Also briefly discussed is the correlation between max-flow min-cut theorem and linear program duality (this book is biased on "combinatorial methods"). Here we would like to point out two interesting but most neglected parts, namely Theorem 5.5 and Section 1.6 (say "Opinion" section). Theorem 5.5, rediscovered several times later, specifies that the existence of a special minimal s-t cut $(X, \overline{X})$ such that $X \subseteq Y$ for each minimal s-t cut $(Y, \overline{Y})$.

Chapter 2 derives a sequence of equivalent results of the max-flow min-cut theorem, including bipartite matching, node-disjoint paths (Menger Theorem), set representatives, and partially ordered sets. All of them are very elegant and useful combinatorial results (e.g. behind the Nobel

Prizes), and its presentation in this systematic way definitely reveals their deep insight (followed by Lovász and Plummer in "Matching Theory"). For the reader, the most helpful way to follow this chapter should be trying to derive these equivalent theorems from each other by herself.

Chapter 3 exhibits the power of max-flow min-cut theorem with real applications. Originally motivated from transportation problems, the authors found a large amount of applications of the network flow model in other areas. Several problems here "might not appear to be" a flow problem at the first sight, but the authors managed to formulate them into one and dispose of as such. This chapter contains the largest number of references, more than the total number of references of the other three chapters combined. Since the focus is laid on the formulations, and the no significantly extra computational effort is required, this chapter might of the least interest for computer scientists.

Chapter 4 is a paraphrase of the (now also classic) Gomory-Hu tree, which is a compact representation of all pairs min s-t cut in the graph. In essence, it studies such a simple question set in undirected graphs: How many invocations of minimal cut algorithms are required to compute the minimal cut of *every* pair of vertices. The trivial answer is $n(n-1)/2$, but Gomory and Hu observed that $(n-1)$ will suffice.

3 Opinion

This book is for people who are interested in the elements of the theory of network flows. All the materials remain definitive, and their exposition is easy to follow; we inherit most of our terminologies from this book (an exception instead of a rule).

To show how forward-looking the authors were when deciding the terminologies, let us have one more thought about the definition of "cuts", the most important concept of this book. The authors explained in Section 1.6 the reason of formulating the cut as a *partition*, instead of a *disconnecting set*. With the benefit of hindsight, we now can see the great advantage of the partition based definition: it easily extends to vertex cuts (also called separators), and more importantly, generalizes to submodular functions.

But the readers must be warned on the efficiency of algorithms, which come without time analysis, and the most important algorithm (for constructing max-flow) is later shown to be not polynomial bounded (in the worst case). This is kind of ironic since the authors emphasize "constructive procedures" that are "computationally effective". This is perfectly understandable since, at the time, complexity was not yet fully defined. The notion of worst case we use to argue its "inefficiency" is too bizarre to occur to the authors who were dealing with real applications. Moreover, the algorithm found in most algorithm books is only slightly adjusted from the one presented here.

For a more comprehensive and (not that much) updated knowledge of the theory of network flows, especially its interaction with linear programming, we would like to recommend Ahuja et al. [1], which is now widely used as a textbook in operations research. For extensions and generalizations of cuts, e.g. vertex cuts, minimum cuts among all pairs, and submodular functions, especially from the algorithmic aspect, the best reference should be the new monograph of Nagamochi and Ibaraki [2].

We would like to close this review with the exciting news on algorithmic aspect of flows: Orlin's recently announced algorithm (`http://jorlin.scripts.mit.edu/Max_flows_in_O(nm)_time.html`).

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice hall, Princeton, New Jersey, 1993.

[2] Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, United Kingdom, 2008.

Review of[10]
Quantum Computing - A Gentle Introduction
by Eleanor Rieffel and Wolfgang Polak
The MIT Press, 2011, Hardcover, $24.00
xiii+372 pages, ISBN: 978-0-262-01506-6 (Cloth, £31.95)

Review by
Kyriakos N. Sgarbas (sgarbas@upatras.gr)
Electrical & Computer Engineering Department,
University of Patras, Hellas (Greece)

1 Overview

This is an introductory book on quantum computation. Since many books on quantum computation have already been reviewed and presented in this column (e.g. issues 34-3, 36-3, 38-1, 40-4, 41-3), I presume that most readers already know what a quantum computer is, or what Grover's algorithm is famous for, so I shall try not to repeat statements of common knowledge. Let me just say that this book addresses the circuit model of quantum computation, where computation is preformed by quantum gates (a.k.a. unitary transformations) and algorithmic processes are usually expressed as circuits built using such gates. Also, the content focuses on quantum algorithmic processes, i.e. it does not contain details on hardware implementations of quantum computers.

2 Summary of Contents

The book begins with a 6-page Table of Contents and a 3-page Preface where the authors define the subject and the scope of the book. The main content is organized in 3 Parts, 13 Chapters and 2 Appendices as follows:

Chapter 1 *"Introduction"*

The introduction defines the scientific area of quantum information processing tracing its origins to the merging of information theory and quantum mechanics. It also defines its main sub-fields: quantum computing, quantum cryptography, quantum communications, and quantum games. It presents a brief history of the area since the early 1980s mentioning the most influential researchers and their findings. (5 pages)

Part I - "Quantum Building Blocks"

Part I of the book includes chapters 2-6 and discusses the main building blocks of quantum information processing, i.e. qubits and quantum gates. Along the theoretical notions of quantum measurement and entanglement, some applications are introduced early on, including quantum key distribution, quantum teleportation and superdense coding. Chapter 6 concerns algorithms, but since they are actually classical ones, this chapter was left in Part I.

Chapter 2 *"Single-Qubit Quantum Systems"*

This chapter illustrates an experiment with polarized photons and describes their properties in order to define what qubits are and explain how they behave under measurement. Dirac's bra/ket notation for quantum states is introduced. The first application is presented here, quite early in the book, and it's the quantum cryptographic *BB84* [2] key distribution protocol. The chapter closes with three representation methods for the state-space of a single-qubit system, namely, algebraic notation, the *extended complex plane*, and *Bloch sphere*. (22 pages)

Chapter 3 *"Multi-Qubit Systems"*

The state-space representation extends to multiple qubit systems in this chapter, after a brief explanation of direct sums and tensor products of vector spaces. The definition and properties of entangled states are discussed along with the basics of multi-qubit measurement. The chapter concludes with another quantum cryptography example, the *Ekert-91 protocol* [6] for quantum key distribution with entangled states. (16 pages)

Chapter 4 *"Measurement of Multi-Qubit States"*

This chapter extends the Dirac bra/ket notation introduced in Chapter 2 to linear transformations and uses it to define projection operators for measurement of multi-qubit systems. Moreover, Hermitian operators are introduced and their eigenspace decomposition is described. Their use in measurement is explained with several examples. The chapter concludes with a description of the *EPR paradox* and *Bell's theorem*. (24 pages)

Chapter 5 *"Quantum State Transformations"*

This chapter considers what happens in a quantum state when it is *not* measured, i.e. the transformations of a closed quantum system. It introduces the notion of a *unitary transformation* (in other words a *quantum gate*) and the corresponding matrix. After a brief presentation of the no-cloning principle, it introduces some simple quantum gates (Pauli I, X, Y, Z, Hadamard, and CNOT) combined in a few simple circuits. Two applications are described here: *(super)dense coding*, and *quantum teleportation*. The chapter explains how quantum gates can be decomposed into circuits made from primitive gates, and concludes with a discussion on the standard circuit model for quantum computation. (27 pages)

Chapter 6 *"Quantum Versions of Classical Computations"*

Every operation performed by a classical computational system can also be performed by a quantum computational system. This chapter describes how to implement such quantum versions of classical computations. It introduces the notion of *reversible classical gates*, shows their analogy to quantum gates and then proves that all classical boolean logic gates can be represented as classical reversible gates, and therefore also as quantum ones. Furthermore, it defines a formal language for the description of quantum circuits and uses this language to express some example quantum circuits performing basic arithmetic operations. (23 pages)

Part II - "Quantum Algorithms"

Including Chapters 7-9, Part II is devoted to the basic quantum algorithms. Starting from the simpler ones (Deutsch's, Simon's) and gradually building up to quantum Fourier transform, Shor's factorization algorithm and Grover's search algorithm.

Chapter 7 *"Introduction to Quantum Algorithms"*

This chapter starts with a general discussion on computing with superpositions and introduces the notion of quantum parallelism. Afterwards, it defines various notions of complexity, including *circuit complexity*, *query complexity* and *communication complexity*. Then it presents *Deutsch's algorithm*, *Deutsch-Jozsa*, *Bernstein-Vazirani*, and *Simon's* algorithms, and (in a special section) *quantum Fourier transform* (QFT). Finally, based on the definition of quantum complexity it describes relations between quantum complexity classes and classic complexity classes. (37 pages)

Chapter 8 *"Shor's Algorithm"*

After a very brief explanation on how the problem of factorization relates to the problem of finding the period of a periodic function, the chapter presents *Shor's algorithm*, gives an example of its operation and analyzes its efficiency. It also presents a variation of Shor's algorithm, where the measurement of the second quantum register can be omitted. The chapter concludes with the presentation of the *discrete logarithm problem* and the *hidden subgroup problem* (including Abelian groups). (14 pages)

Chapter 9 *"Grover's Algorithms and Generalizations"*

This chapter presents a detailed explanation of *Grover's algorithm*, including an outline, its setup, determination of the iteration step, and calculation of the number of iterations. It proves that its performance is indeed optimal (i.e. it is as good as any possible quantum algorithm for the same task). Moreover, it regards *amplitude amplification* as a generalization of Grover's algorithm and uses geometric interpretation to further explain its function. Two methods for "de-randomizing" Grover's algorithm while retaining its efficiency are presented. Both are based on the modification of the rotation angle of the basic algorithm.

Finally, two more variations are discussed, when the number of solutions is unknown: one based on the repeated use of the main algorithm, and the well-known QFT-based *quantum counting* [9]. (26 pages)

Part III - "Entangled Subsystems and Robust Quantum Computation"

Part III includes the final Chapters 10-13 on entangled subsystems, quantum error correction and fault tolerance. Chapter 13 does not fit well in this part since it functions as a binder for several short articles on various topics of quantum computation.

Chapter 10 *"Quantum Subsystems and Properties of Entangled States"*

This chapter defines *density operators* and explains how they are used to model mixed states of quantum subsystems (exhibiting bipartite and multipartite entanglement) and how they are used to model measurements in such systems. It also explains the notion of *superoperators* (the most general class of operators) and how they can be used to express the dynamics of quantum subsystems. The chapter concludes with a discussion of *decoherence*. (40 pages)

Chapter 11 *"Quantum Error Correction"*

The chapter begins with three lengthy examples, each one exhibiting a different simple error correcting code for single-qubit errors: one for bit-flip errors, one for phase-flip errors and one for any type of errors. Then it introduces a general framework for linear quantum error correcting codes, specifying the properties that all such codes should satisfy, and provides some examples including the *[3, 1] repetition code*, the *[7, 4] Hamming code*, and the *Shor code*. Two special sections are devoted to *CSS codes* [11] and *stabilizer codes* [7] respectively. (48 pages)

Chapter 12 *"Fault Tolerance and Robust Quantum Computing"*

This chapter extends quantum error correction with fault-tolerant procedures that enable the error correction to work even when carried out imperfectly. In a lengthy example it demonstrates how *Steane's 7-qubit code* [11] can become fault-tolerant. It then describes *concatenated coding*, a process where a circuit is iteratively replaced by a larger and more robust one. This way a polynomial increase in resources achieves an exponential increase in accuracy, thus leading to a threshold result. The chapter concludes with a threshold theorem ensuring that while the error is below some threshold, arbitrarily long computations can be performed to obtain arbitrarily high accuracy. (18 pages)

Chapter 13 *"Further Topics in Quantum Information Processing"*

This chapter is a collection of ten very brief sections on miscellaneous topics. They mention some more recent quantum algorithms (including *quantum random walks* [4] and *quantum learning theory* [10]), a few more techniques for robust quantum computation, the alternatives to the circuit model of quantum computation (i.e. *measurement-based cluster state*, *adiabatic*, *holonomic*, and *topological quantum computation*), more protocols for quantum cryptography, and some implementation approaches (after presenting *DiVincezzo's criteria* [5]). They also discuss the limitations of quantum computing, some classical algorithmic results obtained from a quantum information processing viewpoint, the simulation of quantum systems, and some more philosophical aspects like the origin of quantum computation power, and what if quantum mechanics is not quite correct. This chapter does not have separate sections with references or exercises, but several references are mentioned throughout the text. (18 pages)

Appendix A *"Some Relations Between Quantum Mechanics and Probability Theory"*

This appendix provides an alternative view of quantum mechanics as a generalization of probability theory. It explains how tensor products can be used for the calculation of probability distributions, how the latter can be expressed with matrices, and how pure and mixed quantum states generalize the classical notion of pure and mixed probability distributions. It concludes with a nice table of analogs and key differences between classical probability and quantum mechanics and an interesting paragraph with references to the bibliography. (10 pages)

Appendix B *"Solving the Abelian Hidden Subgroup Problem"*

The *Abelian hidden subgroup problem* (previously referenced in Chapter 8 with Shor's algorithm) is defined here explicitly and a complete solution is presented using a generalization of Shor's algorithm. In the process, *Schur's lemma* and quantum Fourier transforms over finite Abelian groups are explained. Finally, Simon's and Shor's algorithms are presented as instances of the Abelian hidden subgroup problem. (12 pages)

Every chapter (except 1 and 13) has a section with references to bibliography and a section with exercises (appendices too!). The book concludes with a 11-page bibliography (295 references) and two indices: a Notation Index (3 pages) and a Subject Index (4 pages).

3 Opinion

As its sub-title clearly states, this book aims to be introductory. Moreover, the authors explicitly mention in the Preface that: *"The intent of this book is to make quantum computing accessible to a wide audience of computer scientists, engineers, mathematicians, and anyone with a general interest in the subject who knows sufficient mathematics."* So, when I started reading it, I was somewhat prejudged; I expected to read an introductory book, and I had to review it as an introductory book. So my opinion might be biased in this sense.

From this perspective, the book succeeds in many ways:

It builds up the reader's interest from the beginning by introducing very early some attractive applications (quantum key distribution and quantum teleportation) that traditionally (i.e. in other books) are discussed much later, after many theoretic topics.

It does not use more mathematics than necessary. Of course it needs a good background in linear algebra, and some familiarity with group theory in certain sections that the authors clearly mention in the Preface, but nothing more. For instance, it does not feed the reader with all the details of Hilbert spaces and their properties, just to explain how quantum circuits operate. And I think this is good. Also, no quantum physics background is required. And I think this is even better.

Moreover, throughout the book the authors make a good effort to explain issues and when possible to give insights on how things work instead of just providing lengthy mathematical proofs. The language used is very approachable, simple and direct. In my opinion this book is much easier to read (and much more enjoyable) than the classic Nielsen & Chuang [9] book, but the comparison might not be fair since the latter cannot be considered an introductory book.

From the same perspective, the book seems to miss some details, since it deviates from the standard norm of an introductory quantum computing book in a few unexpected ways. This is not necessarily bad, but a reader who will use it for a first contact with quantum computation will not be able to realize these minor deviations until he reads other books on the same subject and these issues start to rise one by one as small sources of confusion. I shall give a couple of examples:

The most obvious one is the non-standard notation for the CNOT gates in the diagrams. As far as I know there is no ISO/ANSI standard for quantum gate notation, but most contemporary books use the notation developed by Barenco et al. [1], and an introductory book should take this into account. Otherwise its readers might get confused when reading their second book.

The second example concerns amplitude amplification. I believe that most books regard it as a process that some quantum algorithms employ. A more philosophical point of view could even regard it as an inherent property of quantum systems (e.g. like quantum parallelism or entanglement) that a family of quantum algorithms exploit. Presenting it as a generalization of Grover's algorithm is another point of view, which I found strange for an introductory book.

Finally, in some cases introductory and advanced sections seem to mix in the same chapters. This is more evident in the chapters of quantum algorithms and error correction. It might be better in terms of content organization, if all the variations and the modifications of the basic algorithms and codes were explained later in the book, maybe in some appendices. Or at least their sections should be marked as *advanced* or *optional*.

Having said that, I believe that the book succeeds its aim to a great extend and provides an overall interesting, frequently fruitful and occasionally enjoyable reading experience for everyone interested in the subject. It contains all the main topics that one would expect to find in a quantum computation book (and some more), and the sections with the references in each chapter serve as invaluable guides for further reading. Also, the sections of Chapter 13 are very interesting and they could even stand out as stand-alone

articles (if only they were a little bit lengthier). The exercises in each chapter are a big plus if one tries to use it as a textbook. But for use in class, since it lacks of many example solutions, I would recommend combining it with a book with a more systematic solution of exercises, like McMahon's "Quantum Computing Explained" [8].

References

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review Letters, A*, Vol.52, pp.3457-3467, 1995.

[2] C. H. Bennett and G. Brassard. Quantum Cryptography: Public key distribution and coin tossing. *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing*, Bangalore, pp. 175-179, 1984.

[3] G. Brassard, P. Hayer, A. Tapp. *Quantum Counting*, ICALP 1998, pp.820-831.

[4] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, D. A. Spielman. Exponential algorithmic speedup by quantum walk. *Proc. 35th ACM Symposium on Theory of Computing*, pp.59-68, 2003.

[5] D. DiVincenzo. The Physical Implementation of Quantum Computation. *Fortsschritte der Physik*, Vol.48, pp.771-783, 2000.

[6] A. K. Ekert. Quantum cryptography based on Bell's theorem. *Physical Review Letters*, Vol.67, pp.661-663, 1991.

[7] D. Gottesman. Stabilizer Codes and Quantum Error Correction. *Caltech Ph.D. Thesis*, 1997.

[8] D. McMahon, Quantum Computing Explained, Wiley, 2008.

[9] M. A. Nielsen, and I. L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000.

[10] R. A. Servedio. Separating Quantum and Classical Learning. *ICALP 2001*, pp.1065-1080.

[11] A. M. Steane Multiple particle interference and quantum error correction, *Proc. of the Royal Society of London Ser. A*, vol. 452, pp. 2551-2577, 1996.

Review[11] of
The Art of Computer Programming
Volume 4A
by Donald E. Knuth
Pearson Education (Addison-Wesley), 2011
899 pages, Hardcover

Review by
John D. Rogers (jrogers@cs.depaul.edu)
School of Computing, DePaul University

1 Introduction

Early in my days as a graduate student, I had to jump through a set of hoops known as "qualifying exams." These were written tests covering various areas of Computer Science and administered on a Saturday in one of the department's classrooms. On the warm day I took them the classroom happened to face a small quadrangle also bordered by a building with a set of bells. At 9am two dozen keyed-up graduate students began taking the exam. At 9:05 a series of peals rang out across the quad. We all looked up, out the windows, and then at our proctor. Knowing that he was powerless to stop this clangor he looked at us for a moment and then returned to reading his book.

Now it's certainly annoying enough to have bells ringing while one is trying to recall the definition of "Bélády's anomaly" but that wasn't the worst part. As the bells rang I realized that each sequence was a subtle variation on the one before. Wasn't the one just rung identical to the previous one but with a single swap? In fact, hasn't the same bell figured in the last three swaps? Then the real danger struck me. I needed all of my intellectual energy and focus to pass this exam. With difficulty I ignored the bells and their elusive but definite patterns. Amazingly I did so and forged ahead, passing the exam (which put me in place to jump through the next set of hoops, the "oral exams", as in "oral surgery" but a lot less fun).

I soon discovered that we had been the unwilling audience of a performance known as a *change ringing*. But it wasn't until I read Knuth's latest volume in his series on "The Art of Computer Programming" (TAOCP) that I learned how much more that ringing had to do with the Computer Science I enjoy than Bélády and his anomaly.

Change ringing sequences are an example of combinatorial generation: an algorithmic way of listing, one by one and without repetition, each object from a set of combinatorially defined objects. This raises many questions, among them:

- In what order are the objects listed?
- Are there some requirements we want to impose in the transition from one object to the next?
- Is the listing meant to be exhaustive?
- If not exhaustive, is there an ordering that allows the algorithm to skip efficiently over the objects we don't want listed?

A very simple example is listing all bit strings of length n. There is the usual way of listing them lexicographically, starting with all zeroes and ending with all ones, in which case there are transitions from one value to the next requiring the flipping of many bits. If we apply the requirement of flipping exactly one bit for each transition we come to the well-known *Gray code ordering*. For the change ringing mentioned above, the ringers need to generate all permutations on the values 1 through n. Using the *plain changes* sequence requires that each transition swap exactly one pair of adjacent values.

2 Summary

"Combinatorics is the study of the ways in which discrete objects can be arranged into various kinds of patterns." So begins the introduction of this volume's first chapter. Knuth continues with more precise versions of the questions I asked above:

1. Are there any arrangements that conform to a given pattern?
2. Can such an arrangement be found quickly?
3. How many such arrangements are there?
4. Can all such arrangements be visited systematically?
5. Which arrangements optimize the value of some objective function?

This volume proceeds to answer these questions about several kinds of combinatorial objects, including bit strings, boolean expressions, mixed radix numerals, combinations, permutations, and trees.

Continuing the chapter numbering from the first three volumes, this one contains only Chapter 7 (Combinatorial Searching) which, in turn, contains two sections: 7.1 "Zeros and Ones" and 7.2 "Generating All Possibilities". Section 7.1 is now complete but 7.2 currently only contains the sub-section "Generating Basic Combinatorial Patterns"; two sub-sections, one on basic backtracking and one on efficient backtracking, are yet to appear, as are sections 7.3 to 7.10 ([Knu11]).

Most of sub-section 7.2.1 appeared in the fascicles numbers 2, 3, and 4 ([Knu07]) issued a few years ago and reviewed here previously ([Rog08]). It contains:

- 7.2.1.1: Generating all n-tuples
- 7.2.1.2: Generating all permutations
- 7.2.1.3: Generating all combinations
- 7.2.1.4: Generating all partitions
- 7.2.1.5: Generating all set partitions
- 7.2.1.6: Generating all trees
- 7.2.1.7: History and further references

As the titles of the sub-sub-sections indicate Knuth's emphasis is on generating sequences instead of enumerating them. The difference is that, to him, enumeration implies creating a list of all of the objects of a certain size. Instead, his algorithms focus on the transitions: Given a combinatorial object, here is a way to generate the next one. Some of the algorithms take into account an objective function that indicates which objects not to generate.

These algorithms are expressed in pseudo-code and sometimes implemented in the assembly language for Knuth's MMIX machine ([Knu05]).

In each sub-section Knuth provides a thorough treatment of that kind of combinatorial generation. He includes not only the technical and mathematical aspects but also sometimes surprising cultural and historical references. From the history of our own discipline we learn that using a Gray code provided a way to address disk sectors so that at sector boundaries the next sector address is only one bit different than the previous address. Generating all permutations arose in determining the change ringing sequences mentioned above, something a number of writers from the past struggled with. Listing classical Hindi and Greek poetic meters and enumerating the hexagrams of the *I Ching* relied on generating bit sequences. Many puzzles require the solver to determine the right way to traverse a sequence of combinatorial representations of the state of the puzzle.

True to the spirit of puzzle and problem solving, every sub-section ends with many exercises, ranging in difficulty from questions requiring a few seconds reflection to open problems requiring extensive research.

And true to the spirit that these should support and extend the material in the main body of the text there are 300 pages of solutions. Knuth cares passionately about well-crafted exercises and their value in extending and reinforcing the main presentation.

3 Opinion

For readers of this review, there are three ways to view this book. With its extensive development of each topic and the extraordinary number of exercises and their solutions it could certainly serve as a textbook, both in computer science and mathematics. Its list price is $74.99 but less expensive copies can be found.

A second way is to use it as a reference for developing algorithms and code. With the computing power now available it is not unreasonable to attack combinatorial problems by generating and understanding "small" instances, in the hope that this leads to a deeper understanding of the unbounded problem.

The third way is the one I favor most and that I articulated in my earlier review of the fascicles that contained some of this work. This is science writing for computer scientists. There is enough mathematical treatment to stimulate the theoretical parts of our brains and there is enough practical detail to keep our coding neurons firing. But it is the history and references that lift it to being a classic text, one that can send the reader off on a merry chase through time and technique.

The many algorithms presented in the book are (for the most part) written in a way to make it easier to see why the algorithm works. Quite often this means that they are not written to be easy to code. But that really isn't much of an impediment. With a little practice one will find it straightforward to translate his pseudo-code into a working program. In fact, I've found that they work well as coding exercises for CS students learning data structures.

I also enjoy the way puzzles have been worked into the narrative. Knuth has a number of quotes pointing out the importance of puzzle-solving in understanding mathematical principles. In a number of cases combinatorial problems were derived from analyzing a well-known puzzle, such as Ozanam's card arrangement puzzle mentioned on page 3 or the tiring irons or Chinese ring puzzle on page 285.

Knuth rewards a careful reader's attention in other ways as well. Sprinkled throughout the text are pertinent quotes that mostly amuse. For example the last printed page of the section containing exercise answers has this gem: *"I may," said Poirot in a completely unconvinced tone, "be wrong"* ([Chr53]). This is followed on the next page by two quotes, one from the Marx brothers, and how often does that happen in an algorithms text?

This short review cannot even touch on the vast majority of the topics presented in this book. Section 7.1, with the simple title "Zeros and Ones", could be a book on its own, reminding us of the richness of boolean logic and algebra. Section 7.2 has already appeared as a collection of three short books. What's astonishing is that this is the work of a single author...and he's not done yet. As with Agatha Christie's skeptical detective, Knuth might be wrong here and there but his careful scholarship, attention to detail, and obvious love of the subject matter make for a very engaging read.

With respect to this volume's relationship to the first three volumes, it stands for the most part independently. One might argue that volumes 1 through 3 are more fundamental to our discipline but for some, myself included, while I might employ those earlier algorithms more often the ones in this book are the kind I enjoy more. I think the author agrees. In the preface Knuth mentions his inclination to subtitle the volume *The Kind of Programming I Like Best*. His enthusiasm for this material comes through very clearly. It makes one hope that the waiting time to volume 4B is far less than the gap between volumes 3 and 4A.

References

[Chr53] Agatha Christie. *After the Funeral*. 1953.

[Knu05] Donald Knuth. *The Art of Computer Programming, Volume 1, Fascicle 1: MMIX – A RISC Computer for the New Millennium*. Addison-Wesley, 2005.

[Knu07] Donald Knuth. *The Art of Computer Programming (TAOCP), volume 4, fascicles 2, 3, 4*. Addison-Wesley, 2007.

[Knu11] Donald Knuth. The art of computer programming (taocp). `http://http://www-cs-faculty.stanford.edu/\~uno/taocp.html`, 2011.

[Rog08] John Rogers. Review of the Art of Computer Programming, Volume 4, fascicles 2, 3, and 4. *SIGACT News*, September 2008.

Review of[12]
Boolean Function Complexity: Advances and Frontiers
by Stasys Jukna
Springer, 2012
600 pages, $85.00, HARDCOVER

Review by
William Gasarch gasarch@cs.umd.edu
University of Maryland, College Park, 20742

1 Introduction

(**Disclaimer:** I was one of the proofreaders for two chapters of this book.)

In 1986 David S. Johnson published a survey in his NP-Completeness Column called *Computing with One Hand Tied Behind Your Back*. To quote it

... in the circuit complexity model the best lower bound known for a problem in NP has only increased from $2.5n$ *to* $3n$ *since [G&J] (Garey and Johnson) was published seven years ago. Many researchers have thus turned their attention to machine models whose computational power is sufficiently limited that super-polynomial lower bounds may be easier to prove. Such a lower bound would imply only that the problem at hand is hard when your machine has one hand tied behind its back, but the techniques and insights developed might eventually be of use in attacking more general models. In any case the mathematics involved offers its own challenges and rewards. This column will cover three popular ways of restricting general Boolean circuit models and the surprising developments that have recently occurred relating to each.*

The three restrictions he is referring to are monotone circuits, bounded depth circuits, and bounded width branching programs.

This line of research has produced many many results since 1986. So many that its hard (at least for me) to keep track of what lower bounds are known in which models. Fortunately it is not hard for Stasys Jukna! He has written a book that explains (with proofs) most of the progress made in this area.

The book is well written. This is partially because he put the book out on the web asking people to read some of the chapters *ahead of time*. Hence the final product is very polished. Note also that many of the results he is proving were only in conference form with incomplete proofs, or in journals that were behind paywalls, or in Russian. Hence his book does a great service.

Putting that aside, has the program of proving lower bounds on weak models been a success? There are two answers: YES and NO. This depends on what the goals are. There are many results where we have fairly close upper and lower bounds on concrete problems in a simple model of computation. For example, the upper and lower bounds on how many gates you need to compute PARITY with depth k (a constant) match (up to certain factors). We have pinned down the complexity of PARITY in that model! If that is an end in itself then the answer is YES. If it was meant to be a stepping stone to resolving P vs NP then the answer is (so far) NO.

The mathematics used in the book is mostly combinatorics, though clever combinatorics. There are some Fourier transforms and some Linear Algebra; however, there are large parts that can be read without too much background. And the background you do need for the rest isn't that hard to pick up.

The field of Boolean Function Complexity is still quite active. Hence there is a website associated to the book that reports when open problems in the book are solved. It is at

`http://www.thi.informatik.uni-frankfurt.de/~jukna/boolean/index.html`.

2 Summary

The book is in six Parts: The Basics (2 chapters), Communication Complexity (3 chapters), Circuit Complexity (6 chapters), Bounded Depth Circuits (3 chapters), Branching Programs (4 chapters), and Fragments of Proof Complexity (2 chapters). There is also an Epilogue[13] and an Appendix that includes some of the Mathematical background needed.

2.1 Part 1: The Basics

This chapter has a few theorems of interest but is mostly setting up theorems about Boolean Functions to be used later. Here is a theorem of interest that is proven:
(Schnorr 1974) The minimal number of AND and OR gates in an AND-OR-NOT circuit for computing PARITY of n variables is at least $3n-3$.

Is this still the best known? No. There is a table of many known lower bounds at the end of Chapter 1 and it states that the PARITY of n variables is known to take exactly $4n-4$ gates- upper and lower bound, due to Redkin. Johnson's comment above about $3n$ being the best known lower bounds for general circuits is no longer true! Its now $4n$. I do not know whether to be happy or sad.

The theorems stated here to be used later have to do with approximating circuits by polynomials. Most of the theorems are rather technical to state; however, here is one:
(Nisan and Szegedy, 1994) Let f be a Boolean function on n variables. Let $deg_\epsilon(f)$ be the lowest degree of a polynomial $p : \mathsf{R}^n \to \mathsf{R}$ such that, for all $x \in \{0,1\}^n$, $|f(x) - p(x)| \leq \epsilon$. Then $deg_{1/k}(OR_n) = O(\sqrt{n} \ln k)$.

2.2 Part 2: Communication Complexity

There are two ways to view Communication Complexity. Its original motivation was for lower bounds on a certain model for VLSI. It was later used to obtain lower bounds on circuits, branching programs, data structures, and other models. This is all well and good. But questions about communication complexity can also be studied in their own right, without relation to other models. For example, the fact that the communication complexity of equality is exactly $n+1$ is interesting even if it has no application[14] to lower bounds in other models.

This book mostly, but not strictly, takes the view of using Communication Complexity in the service of other models. As such this Part has chapters on *Games on Relations* (used for lower bounds on monotone circuits), *Games on 0-1 Matrices* (large lower bounds on these yield large lower bounds for other models), *Multi-Party Games* (used for lower bounds on streaming algorithms and for Branching programs).

2.3 Part 3: Circuit Complexity

I was surprised to find one of the Parts labeled *Circuit Complexity* since not much is known about lower bounds on general circuits. In fact, this Part has chapters on Formulas (circuits of fan-in 2), Monotone Formulas (no NOT gates), Span Programs, Monotone Circuits, and a more speculative chapter entitled *The Mystery of Negation*. And note that Part 4 is on Bounded Depth Circuits.

We state some theorems of interest.
Let ED_n be the function on $n = 2m \log m$ variables that views the variables as m blocks of $2 \log m$ each, and outputs YES iff all of the blocks are distinct. The number of gates needed in a formula for ED_n is $\Theta(n^2/\log n)$. (In this theorem formulas are fan-out-1 circuits with arbitrary gates. That is, the gates can be any Boolean function of the input).
(Raz and Wigderson 1992) Every monotone circuit for determining if a graph (encoded as $\binom{n}{2}$ boolean variables) has a matching with $n/3$ vertices has depth at least $\Omega(n)$.

[13] the Epilogue looks like its part of the Proof Complexity Chapter, but it is not
[14] My students find this an odd use of the word "application"

2.4 Part 4: Bounded Depth Circuits

The first chapter of this section, Chapter 11, is entirely on depth-3 circuits. This reminds me of the quote that *Getting a PhD means learning more and more about less and less until you've written 400 pages on depth 3 circuits.* To be fair, the book only has around 40 pages on depth 3 circuits. Why are depth 3 circuits interesting? The cynical answer is that we prove things about depth 3 circuits because we can. (Depth 4 seems much harder.) Less cynically, depth 3 is the first interesting level and some of the theorems that were first proven for depth 3 were later proven for general bounded depth circuits. Also, in Section 11.1 the authors notes that high lower bounds for depth three circuits imply superlinear lower bounds for non-monotone log-depth circuits.

As an example of the non-cynical attitude, this book has Hastad's proof that any depth 3 circuit for MAJORITY requires $2^{\Omega(\sqrt{n})}$ gates. It is now known that any depth d circuit for MAJORITY requires $2^{\Omega(1/d)}$. However, the earlier result is interesting for two reasons (1) (minor) for the case of $d = 3$ the earlier results is better, and (2) the proof for just the $d = 3$ case uses no machinery. No Hastad Switching Lemma. It can be taught to High School Student (I've done this.) The proof is very enlightening.

The other chapters are about depth d circuits for constant d. The chapter contains the cornerstone result of the field: PARITY not in constant depth, and also not in constant depth with MOD n gates for n a power of a prime. These were well presented. There are also less well known about circuits with arbitrary gates.

2.5 Part 5: Branching Programs

The first chapter on this section is on Decision Trees which are a special case of branching programs. When I hear *Decision Trees* I think *Sorting requires* $\Omega(n \log n)$ *comparisons* and *ith largest requires* $n + (i-1) \log n$ *comparisons.* Those results are not here. However, the book is on Boolean Function Complexity (Element Distinctness IS discussed) and, once again, if he included everything that people wanted him to include it would double its weight.

This part also has chapters on General Branching Programs, Bounded Replication BP's (e.g., read-once), and Bounded Time. Barrington's theorem is included.

2.6 Fragments of Proof Complexity

This was my personal favorite chapter. It was perfect in terms of material I wanted to learn and had enough background to learn. I learned about Prover-Delayer games and Tree-resolution, regular resolution, (ordinary) resolution, Cutting plane proofs. I have used this material directly in my research. I can't give it a higher compliment than that.

3 Opinion

This book contains everything YOU want to know about Boolean Function Complexity and, at 600 pages, more. Of more importance, its well written and the topics are well considered. It should not just be on the bookshelf of every complexity theorist— it should be on their desk, open to their favorite section.