The Book Review Column[1]
by Frederic Green

Department of Mathematics and Computer Science
Clark University
Worcester, MA 01610
email: fgreen@clarku.edu

In this column, the reviewers review each others' books!

1. **What Can be Computed? A Practical Guide to the Theory of Computation**, by John MacCormick. An appealing introductory text with a somewhat unconventional approach. Review by William Gasarch.

2. **Problems With a Point**, by William Gasarch and Clyde Kruskal. An entertaining, wide-ranging book drawing on Fortnow and Gasarch's long-standing Computational Complexity blog. Review by John MacCormick.

As always, please contact me to write a review; choose from among the books listed on the next pages, or, if you are interested in anything not on the list, just send me a note.

BOOKS THAT NEED REVIEWERS FOR THE SIGACT NEWS COLUMN

Algorithms

1. *Tractability: Practical approach to Hard Problems*, Edited by Bordeaux, Hamadi, Kohli
2. *Recent progress in the Boolean Domain*, Edited by Bernd Steinbach
3. *Network Flow Algorithms,* by David P. Williamson.

Computability, Complexity, Logic

1. *The Foundations of Computability Theory,* by Borut Robič
2. *Applied Logic for Computer Scientists: Computational Deduction and Formal Proofs*, by Mauricio Ayala-Rincón and Flávio L.C. de Moura.
3. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, by Martin Grohe.
4. *Kernelization: Theory of Parameterized Preprocessing,* by Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi.

Miscellaneous Computer Science

1. *Elements of Causal Inference: Foundations and Learning Algorithms*, by Jonas Peters, Dominik Janzing, and Bernhard Schölkopf.
2. *Elements of Parallel Computing*, by Eric Aubanel
3. *CoCo: The colorful history of Tandy's Underdog Computer* by Boisy Pitre and Bill Loguidice
4. *Introduction to Reversible Computing*, by Kalyan S. Perumalla
5. *A Short Course in Computational Geometry and Topology*, by Herbert Edelsbrunner
6. *Partially Observed Markov Decision Processes,* by Vikram Krishnamurthy
7. *Statistical Modeling and Machine Learning for Molecular Biology*, by Alan Moses
8. *The Problem With Software: Why Smart Engineers Write Bad Code,* by Adam Barr.
9. *Language, Cognition, and Computational Models,* Theirry Poibeau and Aline Villavicencio, eds.
10. *Computational Bayesian Statistics, An Introduction,* by M. Antónia Amaral Turkman, Carlos Daniel Paulino, and Peter Müller.
11. *Variational Bayesian Learning Theory,* by Shinichi Nakajima, Kazuho Watanabe, and Masashi Sugiyama.

Cryptography and Security

1. *Cryptography in Constant Parallel Time,* by Benny Appelbaum
2. *A Cryptography Primer: Secrets and Promises*, by Philip N. Klein

Combinatorics and Graph Theory

1. *Finite Geometry and Combinatorial Applications*, by Simeon Ball
2. *Introduction to Random Graphs*, by Alan Frieze and Michał Karoński
3. *Erdős –Ko–Rado Theorems: Algebraic Approaches*, by Christopher Godsil and Karen Meagher
4. *Combinatorics, Words and Symbolic Dynamics,* Edited by Valérie Berthé and Michel Rigo

Miscellaneous Mathematics

1. *Introduction to Probability*, by David F. Anderson, Timo Seppäläinen, and Benedek Valkó.

Review[2] of
What Can Be Computed:
A Practical Guide to the Theory of Computation
by John MacCormick
Princeton University Press, 2018
383 pages, Hardcover, approx $60.00

Review by
William Gasarch (gasarch@cs.umd.edu)
Department of Computer Science
University of Maryland, College Park, MD

1 Introduction

What can be Computed is a textbook for a course that covers Decidable Languages, Computable Functions, Recursively Enumerable Languages, Turing Machines, Reductions, Regular Languages, P, NP, and NP-completeness. Since there are already many books with these topics in them, the question is:

2 Why is This Book Different From All Other Books?

1) **The Ordering of Chapters**

This book has the order:

Decidable Languages, Computable Functions, Recursively Enumerable Languages, Turing Machines, Reductions, Regular Languages, P, NP, and NP-completeness, Original Turing Machine, Godel's Incompleteness Theorem, Karp's 21 Problems.

This ordering is unusual in three ways:

1. Decidable Languages are covered before Regular Languages. Some courses do Regular Languages, Context Free Languages, Decidable Languages, P and NP. Why this order? Because it's the order of discovery. That is a stupid reason. Some courses do Regular Languages, Context Free Languages, P, NP, Decidable Languages. Why this order? Because it's the order of hardness of the languages. That is not a stupid reason, but is it pedagogically best? (I do P and NP before decidable because a reduction of SAT to CLIQUE, two very different languages, makes more sense to the students than a reduction between

$$\{e : \text{ Turing Machine } M_e \text{ halts on some multiple of 42 } \}$$

and

$$\{e : \text{ Turing Machine } M_e \text{ halts on every prime } \}$$

since these are both sets of Turing Machines based on what they do. Plus the reduction itself is done by a Turing Machine.)

[2]

2. Decidable Languages are covered before Turing Machines. Most courses do Turing Machines in order to even *define* decidable languages. Instead this book originally defines Decidability with Python Programs.

3. After the main material there are three chapters that are more historical. I deal with that later in the review.

What is the advantage of this approach? Programming is already familiar and concrete to students taking this course. The book has some material on Python; however, if a student does not know Python they will need to learn it on their own. This will be easy for these awesome millennials and Generation Z-ers.[3]

Note that with Python one can do *everything* in computability theory that you want to do: HALT is undecidable, Rice's theorem, reductions, r.e. languages, and more, can be covered using Python. Once the student is familiar with all of this, then the book talks about Turing Machines. Why do them at all?

1. It is of (some) interest to know the smallest Turing Machine for certain tasks. It is (kind of) interesting (I think but don't care enough to look it up) that Minsky got a Universal Turing Machine down to 5 states. More recently (I think but don't care enough to look it up) Wolfram has offered prize money for questions about small Turing Machines. From my tone you can guess my interest in these kinds of questions. Of more interest (to me) has been the work of Adam Yedidia and Scott Aaronson on small Turing Machines related to non-logic problems in math (see `https://arxiv.org/abs/1605.04343`). However, whatever you think of these kinds of questions, they do not belong in an undergraduate introduction to computability course.

2. The proof of the Cook-Levin Theorem, which does belong in an undergraduate introduction to complexity, does need Turing Machines to prove. Or does it? In this book MacCormick proves the Cook-Levin Theorem using Circuit-SAT and noting that computers come down to AND-OR-NOT gates.

3. We believe the Church-Turing Thesis since the *different* models of computation ended up being equivalent to each other. We believe the poly-Church-Turing Thesis since the *different* models of computation ended up being equivalent to each other with only a poly loss in speed. This is a valid point. But I might just TELL THEM that this is true without defining a single model. That might be cheating.

This thought experiment of teaching computability and complexity without mentioning Turing Machines makes me wonder if the author was not bold enough—he could have left out Turing Machines all together! So why is it in the book at all? For historical reasons. Later in the book is a very nice explanation of the original Turing Machines defined by Turing. Perhaps just having that would be enough.

In summary, I think that the order is fine, though (1) I prefer P and NP and those reductions before computability and those reductions, and (2) I would consider dumping Turing Machines all together.

I recommend the reader of this column use this discussion to think carefully about the order they do use and why.

2) **The Book Does Not Cover Context Free Languages**

Material on this is available in an online appendix. The author wanted to get the book to be for a one-semester course (and note that the cost of the book is low).

Personally I stopped covering context free languages in the course. The mathematical theory of them is ugly (they are not closed under complementation) and the general theory (e.g., theorems like CFL's are in

[3]Future Blog Post Topic: You never hear *Kids were Dumber When I was Growing Up* even though it's true.

P) do not apply to the types of context free languages used in compiler design (e.g, LALR(1)). Also, some modern programming languages are not context free.

I recommend the reader of this column use this discussion to think carefully about why they cover Context Free Languages in their course, and should they.

3) **Parts I, II, and III are** . . .

The 18 chapters are in three parts.

1. Part I: **Computability Theory**. This contains decidability etc, but also regular languages. The title does not match the contents since regular languages should not be under **Computability Theory** but, no harm, no foul.

2. Part II: **Computational Complexity Theory**. This is P, NP, and NP-completeness. There are over 100 pages here. It covers the material very well for the students, though a teacher may get bored reading it.

3. Part III: **Origins and Applications**

   - **The Original Turing Machine** about Turing's definition. This is good for historical prospective, but that may be of more interest to the teacher than the student.
   - **You Can't Prove Everything That's True**: This has a proof of Gödel's Incompleteness theorem. Note that the book assumes the Church-Turing Thesis (that computable = Turing Machines) so the proof is easier to present in this book than it was for Gödel to come up with in 1931. Most books on this material do not want to clutter themselves with **the most important theorem in mathematics in the 20th century**. I can see why (no I can't).
   - **Karp's 21 problems**: This is about the original 21 problems that Karp proved NP-complete. The book does about 5 of the 21 reductions. This could have been earlier, closer to the section on NP-completeness. But no harm, no foul.

3 Opinion

I liked this book. The differences from the usual texts seem good and will make you rethink how you teach the course. On a pragmatic note, there are many good exercises.

Review[4] of
Problems with a Point: Exploring Math and Computer Science
by William Gasarch and Clyde Kruskal
World Scientific, 2019
270 pages, hardcover $68, softcover $37.

Review by
John MacCormick (jmac@dickinson.edu)
Department of Mathematics and Computer Science
Dickinson College, Carlisle, PA

Many readers will be familiar with the *Computational Complexity* blog, started in 2003 by Lance Fortnow and, since 2007, co-blogged by Bill Gasarch. Professor Gasarch is also the first author of *Problems with a Point*, which is based on 24 of Gasarch's *Computational Complexity* blog posts, most of them significantly expanded or reworked with contributions from co-author Clyde Kruskal. The book presents us with 24 stories, problems, or theorems "with a point." Indeed, most chapters have an opening subsection with the pleasingly succinct title "Point," in which the point of the chapter is summarized, implicitly justifying the selection of the corresponding blog post for inclusion in the book. The *Computational Complexity* blog is known for the surprisingly wide ambit of its topics (officially "fun stuff in math and computer science"). This breadth is reflected in the book, which features chapters on a variety of topics, including: the usage of mathematical terms in mainstream publications; what makes a good math competition problem?; what makes a math problem interesting?; what constitutes an elegant proof?; discussion of natural ("sane") reductions; and some healthy lessons about when and how to apply mathematical skepticism and intuition. The accessibility of the selection is admirable: most chapters can be appreciated by a strong high school mathematician, while a few require elementary knowledge of complexity or computability theory.

1 Summary of Contents

The 24 chapters are separated into three parts, roughly corresponding to the level of mathematical maturity needed: Part I, "Stories with a Point"; Part II, "Problems with a Point"; and Part III, "Theorems with a Point."

Part I features ten chapters, most of which relate to the theme of math and theoretical computer science as historical and cultural processes. For example, Chapter 3 has the self-explanatory title "Dispelling the Myth That the Early Logicians Were a Few Axioms Short of a Complete Set." Chapter 5, "How Do Mathematical Objects Get Named?", is a good example of how this book will frequently surprise the reader, delivering insight wrapped in a breezy and jocular style. The authors analyze several methods for naming mathematical objects, including the techniques of "Name the Mathematical Object after Euler or Gauss" and "Hope Someone Names Your Theorem After You." Chapter 10, "Let's Prove Something Instead of Proving That We Can't Prove Something!!!" is more familiar territory for the complexity theory crowd. And, despite being the only chapter that starts with a "Rant" subsection instead of the usual "Point," this is very far indeed from the usual rant about complexity theory being in a rut and needing new approaches. The authors add a provocative twist to this theme, providing a new take on the complexity theory rut issue. And speaking of provocative twists, Chapters 8 and 9—ostensibly on the topic of interdisciplinary Ramsey Theory—deliver another subtle and edifying surprise, which would be ruined by further discussion here.

Part II comprises nine chapters, most of which address the nature of mathematical proof or the nature of mathematical problems. Cutting across these themes are frequent connections to high school math competitions and what they tell us about math more generally. Of the three chapters about proofs, two investigate the question of what makes a proof "elegant," by analyzing proofs of a coloring theorem and a theorem about decimal representations of integers respectively. Another proof-themed chapter discusses a problem related to sums of reciprocals. The problem has many distinct proofs—in some sense, infinitely many. The authors have empirical data about the solutions submitted when the problem appeared in a high school math competition, and this leads to an interesting discussion on the process behind developing proofs. Chapter 17 investigates the methodology for solving a certain communication problem (a so-called "hat" problem), leading to a salutary lesson about when it is the right time to abandon a failed approach and look for a new one. Four chapters investigate the issue of what makes a good math problem, for various specific meanings of "good." For example: Chapter 14 asks what it might mean for a problem to be too hard, by analyzing another sum-of-reciprocals problem; Chapter 18 investigates the circumstances under which a trick question is really just a stupid question; and Chapter 19 employs a problem about multiparty communication complexity to demonstrate how a problem can be made more fun by seeking pleasing solutions rather than optimal ones.

Part III comprises five chapters containing more advanced material on similar themes. Chapter 20 revisits the concept of elegance in proofs, demonstrating via examples from combinatorics how a less elegant proof can sometimes be more convincing. Chapter 21 presents a theorem about perfect numbers that also happens to be true for a more general class of numbers. The authors discuss why the less general form of the theorem is in some ways more interesting. Chapter 23 is another case study in how math is developed, this time examining the rectangle-free coloring of grids. That leaves Chapters 22 and 24, which are discussed next in a little more detail.

Chapter 22 gives a "sane" polynomial-time reduction from 4-colorability to 3-colorability: $\mathrm{COL}_4 \leq \mathrm{COL}_3$. At first glance, this may seem pointless, since both problems are NP-complete and, for any NP-complete problems A and B, we know immediately that $A \leq B$. Moreover, we can explicitly construct a reduction from A to B as follows: (i) use the proof technique in the Cook-Levin theorem to construct a reduction from A to SAT; (ii) examine the proof of B's NP-completeness, from which one can probably construct a reduction from SAT to B. Chaining together these two reductions gives the desired reduction from A to B. But the key point is that this reduction is "insane": it is unnatural, because it goes via a Boolean formula (and I would add that this reduction is almost certainly woefully inefficient, albeit polynomial time). The authors give empirical evidence from teaching a theory class that it can be beneficial to present more natural reductions when possible, and as a specific example they provide an elegant reduction from COL_4 to COL_3.

Chapter 24 begins with a remarkably concise, beautifully written, self-contained introduction to computability theory. The chapter then moves on to its main point, which is to highlight an interesting technique for making conjectures. When faced with a mathematical statement S whose truth value is unknown, we must decide whether to expend more effort on proving S or $\neg S$. Sometimes, one of these alternatives (for concreteness, suppose it is $\neg S$) implies another statement, say T, which is well-studied but whose truth value is also unknown. An obvious case arises when T is the statement "P=NP"; in this case, it is best (unless suffering from severe hubris) to expend effort on proving S rather than $\neg S$. The authors give a more subtle example from Kolmogorov complexity in which T ends up being the statement that there exists a "natural" intermediary set (i.e. a set I defined by some natural property, such that I is uncomputable but strictly easier than the halting problem). The fact that no such set is known leads one to pursue the correct strategy on the original problem in Kolmogorov complexity.

2 Opinion

The genre of books that consist of blog-post collections has been around for a while. In the last decade, Richard Lipton has published two such books[5] (one with co-blogger Ken Regan) based on posts about Gödel's lost letter, and Terence Tao published *Structure and Randomness: Pages from Year One of a Mathematical Blog*[6] in 2008. The genre is really a variant of a much older form: before the era of blogs, it was not uncommon for authors to publish collections of their short pieces. Back in the 1980s, for example, the mathematician Tom Körner published an extraordinary book called *Fourier Analysis*, which (despite the title) includes musings ranging from number theory and numerical analysis to statistics and astronomy.

But blogs and books (as they have traditionally been defined) serve somewhat different purposes, so there is a challenge in tuning the content when adapting from blog to book. Stylistically, the author of a blog post sometimes employs whimsical phraseology and tolerates a few typos, which may seem out of place in a printed book. There are also content-related challenges, such as the varying lengths of the posts, the level of assumed knowledge (which may vary considerably between posts), and the coherence of the topics themselves. The methodology for explanations can also differ, since it's common for blogs to contain hyperlinks or references to external sites which may seem awkward in a traditional book. For example, *Problems with a Point* contains a brief discussion of the Sleeping Beauty Paradox in which the authors write, "The Wikipedia reference is pretty good. We encourage you to go read it."

As this example demonstrates, Gasarch and Kruskal have deliberately aimed for the blog end of the blog-to-book spectrum, frequently spurning the constraints of traditional books. This is a highly successful approach, resulting in a book that is packed with fun, spirited discussion of stimulating topics from mathematics and computer science. As already mentioned, the authors do a fine job of making the content accessible to a mature high school mathematician. A handful of chapters require a background in complexity theory, but these are clearly signposted. The chosen topics cohere well, with only one or two outliers.

Readers expecting conventional prose and explanations may occasionally feel jarred, but I found these moments instead challenged me to think more carefully about how best to reach a modern audience that has access to modern online resources. For example, suppose one is trying to explain, in a lecture or book chapter, a topic for which there already exists a truly excellent Wikipedia article. Under what circumstances is it appropriate to use that article as the primary or only source for the explanation? The instinctive response is that one needs to "add value" by providing one's own explanation, but is it really rational to invest the required effort for this, and would the result actually be improved?

Some of the themes in *Problems with a Point* have of course been explored elsewhere. For meditations on the nature of mathematical proof and the process of solving mathematical problems, there is a rather large canon of competition, including the classic works by Hardy and Polya. But *Problems with a Point* breaks new ground in several ways: the choice of attractive problems with a computer science flavor; a zesty style that frequently challenges readers to participate in the action; several ingenious twists in the narrative that force readers to re-evaluate their ideas; and masterful, surprisingly light, presentations of some rather subtle material. The authors' joy and delight in "fun stuff in math and computer science" sparkle on every page. Anyone who dips into *Problems with a Point* will capture some of that joy for themselves.

[5] Both reviewed in this column by Bill Gasarch, SIGACT News 41(4), 2010 and 45(2), 2014.

[6] Also reviewed here, by Bill Gasarch, SIGACT News 46(2), 2015.