The Book Review Column[1]

by Frederic Green

Department of Mathematics and Computer Science
Clark University
Worcester, MA 02465
email: `fgreen@clarku.edu`

In this column we review these three books:

1. **Compact Data Structures – A Practical Approach**, by Gonzalo Navarro. An unusual approach focussing on data structures that use as little space as is theoretically possible, while maintaining practicality. Review by László Kozma.

2. **Power Up: Unlocking the Hidden Mathematics in Video Games**, by Matthew Lane. A revealing look at the mathematics, physics, and computer science that underlies (good) video games, with special attention to their potential valuable role in education. Review by S. V. Nagaraj.

3. **Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis (Second Edition)**, by Michael Mitzenmacher and Eli Upfal. The second edition[2] of a distinguished text on a rich area that is central to computer and data science. Review by Aravind Srinivasan.

Many thanks to these and all reviewers over the years who have graciously volunteered this valuable service to the community! Interested in reviewing? I hope so. Remember the perks: Learning (more?) about a field that interests you, a review on these pages, and a *free book*! The books listed on the next two pages are available, but the list is not exhaustive: I'm open to suggestions.

[1]© Frederic Green, 2018.
[2]First edition reviewed in SIGACT News Vol. 38, No. 3 (2005).

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. *Finite Elements: Theory and Algorithms*, by Sahikumaar Ganesan and Lutz Tobiska
4. *Introduction to Property Testing*, by Oded Goldreich.

Programming Languages

1. *Practical Foundations for Programming Languages,* by Robert Harper

Miscellaneous Computer Science

1. *Actual Causality*, by Joseph Y. Halpern
2. *Elements of Causal Inference: Foundations and Learning Algorithms*, by Jonas Peters, Dominik Janzing, and Bernhard Schölkopf.
3. *Elements of Parallel Computing*, by Eric Aubanel
4. *CoCo: The colorful history of Tandy's Underdog Computer* by Boisy Pitre and Bill Loguidice
5. *Introduction to Reversible Computing*, by Kalyan S. Perumalla
6. *A Short Course in Computational Geometry and Topology*, by Herbert Edelsbrunner
7. *Partially Observed Markov Decision Processes,* by Vikram Krishnamurthy
8. *Statistical Modeling and Machine Learning for Molecular Biology*, by Alan Moses
9. *Market Design: A Linear Programming Approach to Auctions and Matching,* by Martin Bichler.

Computability, Complexity, Logic

1. *The Foundations of Computability Theory,* by Borut Robič
2. *Models of Computation*, by Roberto Bruni and Ugo Montanari
3. *Proof Analysis: A Contribution to Hilbert's Last Problem* by Negri and Von Plato.
4. *Applied Logic for Computer Scientists: Computational Deduction and Formal Proofs*, by Mauricio Ayala-Rincón and Flávio L.C. de Moura.
5. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, by Martin Grohe.

Cryptography and Security

1. *Cryptography in Constant Parallel Time,* by Benny Appelbaum
2. *Secure Multiparty Computation and Secret Sharing*, Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen
3. *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 and History

1. *Professor Stewart's Casebook of Mathematical Mysteries* by Ian Stewart
2. *Introduction to Probability*, by David F. Anderson, Timo Seppäläinen, and Benedek Valkó.

Review of[3]
Compact Data Structures – a practical approach
by Gonzalo Navarro
Cambridge University Press, 2016
570 pages, Hardcover, $89.99

Review by
László Kozma (`l.kozma@tue.nl`)
TU Eindhoven, Netherlands

1 Introduction and Opinion

The book studies fundamental data structures and representations of combinatorial objects, including arrays, bitvectors, permutations, trees, graphs, etc. There are, of course, a great number of existing textbooks and lecture notes about data structures. The perspective of this book is, however, different from the usual presentations. In some sense, this book starts exactly where others leave off saying "we omit the low-level details of representation", and concerns itself *mostly* with the low-level representation. Specifically, it looks at how to organize and store data structures using as little space as possible while supporting efficient queries and operations. For the considered structures, the book explores the trade-off between reduced space and speed of operation. The analyses of the presented solutions go beyond the usual asymptotics, and also consider exact constant factors (particularly for the leading term of the space complexity); useful practical considerations are also discussed.

The book is a real gem! My excitement while reading it was similar to what I felt when learning to program as a child, browsing the bit-level details in a technical manual of an obscure computer architecture. Even though data structures are my main research interest, much of the material was new to me. There is a remarkable attention to detail, pseudocode in the book is written very carefully, with all details explicitly given. In this sense, the experience of reading the book also reminds me of Knuth's "The Art of Computer Programming". In contrast to TAOCP, however, the results in this book require far less advanced mathematics. Instead, they combine a high number of ingenious tricks for data manipulation. Still, it would be unfair to describe the book as a bag of tricks, as the described techniques build on one another, and come together in coherent designs. The amount of explicit low-level detail makes a lighter skimming of the book slightly difficult, but it pays off when one tries to use the book as a reference, or for implementation.

Most of the techniques and tools described are not difficult to understand in themselves; they are, however, combined into solutions that are often counter-intuitive and surprisingly efficient. Readers may test themselves by attempting to find their own solutions before reading the descriptions of the data structures in the book, in order to appreciate the difficulty of the problems.

The topics covered in the book show a beautiful unity, even though they come from a number of different sources: (1) ideas from information and coding theory, going back to the beginnings of the field (Shannon-Fano/Huffman coding, number representations, etc), (2) results in algorithms and data structures with particular focus on reduced space, going back (at least) to the eighties ("succinct data structures", string representations, etc) – many results of this category in the book are due to the author and his collaborators, (3) techniques and tricks (some of them folklore) from programming practice.

[3]©2018, László Kozma

The computational model the book adopts is essentially the "word RAM" with constant-time operations on words; many of the techniques used in the book would be labeled as "bit-tricks" by the more theoretical side of the community. Far from being apologetic about this, the book embraces bit-tricks with enthusiasm, as an important component of this line of research.

For concreteness, and to give a flavor of the kind of topics studied in the book, let me give two examples.

Consider storing a graph in memory, such as to allow basic traversal and other operations. Two standard representations are adjacency matrices and collections of adjacency lists. The former is clearly inefficient in its memory-usage, especially for sparse graphs. The latter may seem efficient, but for directed graphs it does not easily allow the querying of edges pointing *to* a vertex. A straightforward extension to address this issue would double the space usage, and this is not a decision taken lightly in this book. For undirected graphs, the space usage is already, in some sense, twice what it should be, as edges are stored in the adjacency lists of both of their endpoints.

The book describes two general approaches for improving this situation. One is to store the adjacency matrix row-by-row, using primitives developed for compressed storage of bitvectors. The other more practical solution is to store a concatenated sequence of all adjacency lists, and a bitvector that encodes the number of neighbors of each vertex. The out-neighborhood of a vertex can be computed in a straightforward way. For the in-neighborhood, we need to enumerate occurrences of a vertex in the concatenated adjacency list, and to find out to which original adjacency list a certain occurrence belongs. For these tasks simple arithmetic can be used, on top of efficient primitives for navigating the sequences and bitvectors, developed elsewhere in the book.

The second example I mention asks to store a permutation $\pi : [n] \rightarrow [n]$. As there are $n!$ such permutations, any representation needs at least $\log_2 n! = n \log_2 n + O(n)$ bits. Besides identifying the permutation π, we would also like to efficiently perform some queries, such as computing $\pi(i)$, $\pi^{-1}(i)$, or more generally, $\pi^k(i)$ for some i, k, without using much more storage than the information-theoretic minimum.

Here is an idea: simply store $\pi(i)$ for all i, and to find $\pi^{-1}(i)$, walk along the cycle of i, by considering $i, \pi(i), \pi^2(i), \dots$, until reaching i again (then we have identified $\pi^{-1}(i)$. Unfortunately, a cycle may be of length n, which would be too slow for a query. To speed things up, we store at every t-th (or so) step in every cycle, a shortcut "back-edge" from j to $\pi^{-t}(j)$. This lets us to find $\pi^{-1}(i)$ by doing at most t steps. The smaller the value t the more efficient the search, but the amount of space used also increases accordingly. To find any $\pi^k(i)$ efficiently, a slightly more elaborate scheme is described based on similar ideas. The exact details of how we store cycles and back-edges is of course the interesting part of the scheme, and here the solution again builds on primitives described in other parts of the book. Finally, for permutations with special structure (e.g. long runs of monotone entries) a more space-efficient, compressed representation is presented.

In 540 pages, the book discusses a large number of data structuring questions, exploring various designs and tradeoffs, followed by relevant applications. The book is very well-organized, and its concept is compelling (once the reader gets the hang of it), the various results fit together and build upon each other. I imagine that seminars could easily be based on the book; it can also be useful for self-study and reference, even for practitioners. The summaries and bibliographic notes at the end of each chapter are very helpful; the fact that several of the references are from the last few years indicate that the field is very active and the book can be used as a starting point for research. Some of the solutions may not be the last word on the topic, in terms of performance or simplicity. (The last chapter gives a broader selection of topics that the author considers indicative of the future of the field.)

Throughout the book, the explanations are complemented by well-chosen examples. Additionally, in every chapter there are several longer, worked-out example applications that motivate the data structures. These cover a wide variety of domains (some random examples: grammar-compression, routing, inverted indexes, range queries, LZ78 compression) and significantly broaden the scope of the book. These applications are focused on concrete, well-defined tasks; the book also makes one curious about complex real-world system, e.g. database engines, file systems, web servers, etc.: which bitvector-, sequence-, tree-, graph-, etc. representations are most useful and actually used in such applications? What are the practical, low-level requirements in such systems?

A minor comment could be that although hashing is briefly discussed in the book, hardly any good applications of it are mentioned. From the title, I expected that part of the book would be about hashing-based space-efficient data structures (e.g. Bloom-filters or approximate counting such as HyperLogLog). These are both *compact* and *practical*, would thus seem to fit with the stated purpose of the book. The omission of such topics is understandable, however, as the book focuses only on *exact* data structures (and contains a significant amount of material as it is). Finally, *compactness* of data structures is only one facet of their practicality, often at odds with others, such as *locality of reference* (as discussed in the last chapter). Whether the proposed data structures play along nicely with modern computer architectures and applications in terms of other criteria may be interesting to further discuss.

2 Summary of the book

1. Introduction. Discussion of the context, motivation, and basic notation.
2. Entropy and Coding. Main topics are: entropy, Shannon-Fano/Huffman coding, variable-length coding of integers.
3. Arrays. Arrays store a sequence of items (typically numbers), allowing access to an item at an arbitrary location. If all items are stored on the same number of bits, then the representation is straightforward. If items may have different lengths (to avoid wasting space), finding the location at which a certain item starts is trickier. Several alternatives are considered for a secondary structure that allows fast access in this case: sampled pointers, dense pointers, direct access codes, Elias-Fano codes. Computation of prefix sums, finding the first index where the prefix sum exceeds some value, and initializable arrays are discussed as applications.
4. Bitvectors. Bitvectors store an array of bits, allowing the reading of the i-th bit, i.e. *access* (no write-operations for now), as well as *rank* and *select* queries. Rank computes the number of 0s (or 1s) up to a certain index, and select computes the index of the i-th 0 (or 1). Rank and select turn out to be crucial components of most compact data structures studied in the book. A key question is how to take advantage of structure in the bitvector, i.e. how to compress it such that the total number of bits used is not far from the various entropy-measures, while supporting the operations efficiently. Perhaps surprisingly, under various assumptions, bitvectors can be stored in a compressed form, while supporting constant-cost operations (in the RAM model), at the cost of a slightly sublinear space-overhead. Predecessor/successor queries, dictionaries, sets, and perfect hashing are discussed as applications.
5. Permutations. How to store permutations compactly, such as to support queries on predecessors, successors, and arbitrary powers of an element. As mentioned in the example, various schemes based on cycles and shortcut-edges are discussed, using bitvectors and rank/select operations in their implementation. In the end, the cost of queries is $O(1/\varepsilon)$, with space roughly $(1 + \varepsilon)$ times the information-theoretic optimum. More compressed representations are discussed for permutations with long runs.

6. Sequences. Sequences (strings) generalize bitvectors to a larger alphabet. The same operations, access, rank, and select are supported. Trivial extensions of bitvectors would increase the space requirement by a factor roughly the size of the alphabet. More sophisticated solutions reduce this factor to a logarithm of the alphabet size. Ultimately, select is supported in constant time, access/rank are supported with a $\log\log(\cdot)$ dependence on the alphabet size, while maintaining close to optimal space.

Further compressed schemes are also presented and the sophisticated wavelet tree structure is described. Wavelet trees store, informally, a recursive decomposition of a sequence into subsequences (not necessarily contiguous). Each node of the wavelet tree stores a subsequence, encoded using bitvectors. Here, wavelet trees are used for fast rank/select, but other applications are also mentioned. Various extensions and compressed variants are also described. Overall I found this section and the next two, perhaps the technically most interesting parts of the book.

7. Parentheses. Sequences consisting of an equal number of "(" and ")" characters such that no prefix contains more ")" than "(" can be viewed as balanced systems of parentheses. Natural queries include finding the matching closing or opening pair of a parenthesis, finding the level of nesting or the innermost containing pair of parentheses at a certain position, and other, more complex questions.

To support all queries in logarithmic time, with at most a linear number of extra bits on top of the parentheses-sequence itself, a simple but somewhat counter-intuitive solution is presented using range min-max trees (a kind of augmented, level-linked, balanced binary search tree). Then, an improvement to $O(\log\log n)$ query times is presented, mentioning (theoretically) even more efficient solutions in the literature; I found this intriguing and felt curious to read more. The $O(\log\log n)$-construction already combines several nice ideas: a hierarchical search structure built on top of the previous solution, a left-to-right-minimum tree (closely related to Cartesian trees), and additional techniques for level-ancestor and range-minimum queries. While I was familiar with some of these concepts and structures, I found their combined application fascinating.

8. Trees. The goal here is to represent rooted trees (not necessarily binary) using a number of bits close to the information-theoretic optimum $\sim 2n$ (as opposed to the $\Omega(n \log n)$ bits used in a naive representation), supporting traversal from a node to its parent, first/last child, or sibling, and finding the depth, height, subtree-size of a node, as well as more exotic queries that may come up in applications. The two main solutions presented are the LOUDS-scheme, and a scheme that uses strings of parentheses (as studied in the previous chapter). The first scheme encodes for each node, level-by-level, its number of children in unary (as a sequence of ones ended by a zero). The total number of bits of $2n$ is clear, as each node contributes a one to its parent, and a zero to itself. The implementation of tree operations is less obvious, and for this purpose, again, the bitvector operations developed earlier are used (adding some small space-overhead). In the second scheme, a preorder-traversal (depth-first-search) is performed, and encoded as a balanced parenthesis-string (opening and closing for the first, resp. last visit of a node). Tree operations are then implemented using the parentheses-operations developed earlier. Overall, most operations are implemented in near-constant or even constant time (at the cost of using the most complicated components, with some additional space overhead). Many applications of trees are described.

9. Graphs. Similarly to trees, the goal is to represent graphs, such as to allow queries about neighbors and traversal from one vertex to another. The straightforward adjacency list and matrix representations are discussed, and more advanced schemes are given that improve their functionality and space usage. For certain special graph families (clustered graphs, k-page, planar graphs) further space-saving is obtained that gets close to the information-theoretic bounds. Particularly for planar graphs and triangulations, interesting graph-theoretic observations are used to construct a certain canonical spanning tree on which the solution is based. Compact storage of graphs is of course crucial in many modern applications, of which a few are

mentioned.
10. Grids. The grid data structure stores points in a rectangular integer grid, allowing queries, such as the number of points within an axis-parallel query rectangle. Grids have several applications in geometry, text-searching and others. The first solution transforms the point set into a sequence (by perturbing points that are aligned on the same vertical line) and stores the sequence using the wavelet tree structure introduced earlier. This "geometric view" makes the intuition and motivation behind wavelet trees clearer. A different tree-based representation is also described.
11. Texts. This important chapter discusses both classical results and newer developments. Texts (strings) are essentially the same as sequences, except that here the main operation of interest is to search for occurrences of one string in another. The important suffix array and suffix tree structures are discussed, together with their compressed representation. Further topics include the FM-index and the Burrows-Wheeler transform.
12. Dynamic structures. This chapter discusses how and to what extent the structures described in the book can be made dynamic (i.e. to allow updates, typically the insertion and deletion of items). While dynamism is possible in most cases (with considerable difficulty) it involves a moderate space-penalty and a not-so-moderate increase in the running time of the operations.
13. Recent trends. These include *encoding* data structures (these save space by allowing certain queries, but not necessarily the reconstruction of the actual data), data structures that attempt to compress based on higher order regularities (especially for text data), and issues related to memory hierarchies and locality of reference.

Overall, the book is a wonderful resource for anyone interested in data structures from a practical point of view, particularly concerning space-efficiency. It collects a large body of work previously scattered in articles and fills a gap in the literature. I am happy to have read it and will surely revisit parts of it more carefully in the future.

Review of[4]
Power Up: Unlocking the hidden mathematics in video games
Matthew Lane
Princeton University Press, 2017
264 pages, Hardback, \$29.95; eBook, \$24.73

Review by
S. V. Nagaraj svnagaraj@acm.org
VIT, Chennai Campus, India

1 Introduction

Video games provide entertainment to those who play them. This book looks at the hidden mathematics behind video games. It is meant for all those who love playing video games, so that they may explore mathematical ideas in the games. It is also aimed at mathematics teachers for inspiring their students, and for all those who love mathematics and games. Even those who hate mathematics but love video games are also a part of the target audience of this book. However, this particular category of readers needs at least some familiarity with high school mathematics to benefit from the book. The author states that many people enjoy playing computer / video games but few would be indulging themselves in doing some math. He terms this a consequence of poor marketing of mathematics since time immemorial. The main purpose of the book is to highlight that mathematics can be fun and that it can be found even in the most unbelievable or unexpected places such as in computer games.

2 Summary

The book consists of an introductory section followed by nine chapters, notes, bibliography and an index. In the introduction, we come to know about video games such as *Super Mario Bros.*, *The Legend of Zelda*, and *Metroid*. Lane brings mathematics into the picture as he talks about the proof by mathematician Andrew Wiles of Fermat's Last Theorem. Wiles solved the longstanding problem of Fermat by showing that the equation $x^n + y^n = z^n$ has no solutions when x, y, z and n are whole numbers and where n is greater than 2. The author mentions that there are many similarities in the exploration involved in mathematics with even those in games such as *Zelda* and *Metroid*. He observes with sadness that many educational math games have neither educational value nor are they great enough to be called as games.

Chapter 1 [Let's get physical] discusses the learning from video games about the physical world. Lane starts by recollecting physics classes in his high school days. He looks at questions such as

If you go skydiving, what will your trajectory look like after you step out of the plane but before you open up your parachute? (Ignore air resistance.)

Surprisingly, he blames games such as *Super Mario Bros.* for being responsible for his lack of understanding of basic physics. The author looks at several other questions and stresses that *Super Mario Bros.*

[4]©2018, S.V.Nagaraj

was not the only one to ignore the fundamental laws of physics, in fact an entire genre of games did so. Lane mentions the genre of platformers which describe any game in which jumping from platform to platform is an essential component of the gameplay. Unfortunately, most of the time this genre of games and even others had little or no regard for realistic physics. I agree with Lane and note that many video games which did not respect the laws of physics were not well received. As an example, I note a soccer video game *FIFA 14* that initially did not portray shots in a realistic manner.

On the basis of his own experience, and by studying several video games, the author goes on to emphasize that this inaccurate portrayal of physics in games does indeed have an adverse effect on learning. Lane observes that even recent games do not have realistic jumps and do not take into account crucial factors such as gravity. He makes a very important pronouncement: video games should help in driving away misconceptions rather than reinforcing them. I agree, but wonder if game designers really pay heed to this. Now comes the big question: Can physics be really learnt the right way through gameplay? The author speaks about the realistic physics engine of the game *LittleBigPlanet*. He mentions contests held using *LittleBigPlanet* to create content that helped students learn science and math. He opines that the game *Portal* has more educational value.

Lane poses a challenging question: *How about developing games that would allow for deep and rich exploration outside of our everyday experience but that would also more accurately reflect the natural laws of our universe?* An interesting game discussed in this context is *A Slower Speed of Light* which attempts to make Einstein's theory of special relativity seem more intuitive. *Miegakure* is an engaging game that helps to teach the fourth dimension. The author concludes by saying that game design always has educational consequences, because players are always learning something when they play. Furthermore, realism is not necessarily a requirement for a game to be used as an educational resource. The game *Minecraft* is discussed and Lane then poses another tough question: *Can games be useful educational resources?* He opines that they can be, although the process of extracting educational value out of games is not always very straightforward.

In Chapter 2 [Repeat offenders] Lane recalls his experience regarding game shows on TV. *Family Feud* is one such game show. Teams compete for points by trying to guess the most popular responses to survey questions from a sample. Lane discovered a video game version of *Family Feud* in a store. He highlights a serious problem with these games: the number of questions that are asked is fixed, and as you play the game, you are likely to bump into questions you have already seen. The more you play, oftentimes these duplicates will come up. Lane gets into probability and statistics and looks at the frequency of repetition. The negative impact of repetitive questions is shown by noting how the popularity of the game *Draw Something* plummeted in a short span of time and how the firm that had invested huge sums of money lost possibly millions of dollars. This highlights that even game designers have to know some math if their game should be successful. Approaches to deal with the problem of repetition are then discussed mathematically.

Chapter 3 [Get out the voting system] introduces the reader to voting. In the game *Everybody Votes*, the players are given just two options to choose from. For example, for the question: *Where would you rather live?*, the options were just beach or mountain. A critic complained that this was inadequate and that someone might perhaps wish to live in a cave, the wilderness, a city, under the sea or in a van down by the river. Lane mentions that someone might get easily carried away by the complaint and overlook the problems involved in allowing more than two alternatives. For brevity, we look at just one of those questions that must be answered. For example, if we offer more than two options, what determines which option wins? Lane stresses that the answers to these questions are not simple enough. He points out an interesting fact: some voting systems can unnecessarily complicate an election, and as a matter of fact, the winner of an election is on certain occasions decided more by the rules of the system than by voters' real preferences! Lane looks at video games which fortunately provide us with lots of examples of voting systems.

Lane looks at plurality voting and its shortcomings. In this type of voting, every voter casts his or her vote for his or her preference. After everyone has cast their vote, the votes are counted, and the winner is the one who got the bulk of the votes. When there are just two options for the voters to choose from, some of the flaws of plurality voting fade away but when there are more options the drawbacks in plurality voting become apparent. Ranked-choice voting systems including Borda count and instant runoff voting are then explored by Lane who poses a thought provoking question to the reader:

After all, if we can get an entirely different winner by changing the voting rules, what's the point of having a vote in the first place?

The impossibility theorem of economist Kenneth Arrow is then introduced and it is said that score voting and approval voting avoid the clutches of Arrow's impossibility theorem. An interesting section tells us what game developers know that politicians don't. The developers of *Assassin's Creed IV: Black Flag* decided to implement score voting to get feedback from players. The game *LittleBigPlanet 2* also initially used a form of score voting and later approval voting to get feedback from players. Lane ushers in the concept of the Wilson score confidence intervals while discussing feedback. We come across a nice valid point made by Lane: *An added sprinkle of mathematical thinking could, at the very least, help make future games even better.*

Chapter 4 [Knowing the score] looks at scoring in games. Often, players like to compare their scores. However, there is a catch here: if you just know someone's score, there is often no way to know with certainty how the player actually achieved that score. While discussing the nice game *Sunny Day Sky*, where a bear gets to use umbrellas to fly over cars, Lane introduces the partition function which has been studied in depth by number theorists. The number of ways we can express a whole number as a sum of squares is also looked at. Figuring out the details of how someone played from just a few pieces of information is not really feasible. However, in this search we often encounter some beautiful mathematics. Lane says that mobile gaming has several examples of interesting scoring mechanics and we could fill an entire book by exploring them. To illustrate this, he looks in detail at the scoring mechanism in the game *Threes!* It is also shown how math can help in detecting fake scores. The concept of measuring distance between two pieces of text is introduced through the password guessing mini game in *Fallout 4*. The player is presented with a list of possible passwords to hack a computer. The goal is to find the password in as few guesses as possible. Lane looks at strategies any good mathematician would adopt for achieving this. He emphasizes the fact that measuring the distance between two pieces of text has applications in biology and computer science as well. Logistic curves are also briefly discussed in the chapter.

Chapter 5 [The thrill of the chase] starts with the game *Mario Kart 64* which allows shells of two kinds: green and red to attack opponents. Lane looks at effective ways to use these shells. He studies two different shooting strategies: attacking head-on and banking shots off of a single wall. After doing some math, Lane concludes that if you have got a green shell then it is not really worth strategizing too much. This is primarily due to the difficulty in calculating angles in the heat of battle, and also since it is not easy to anticipate the opponents' future moves. It is probably best to fire at close range or hunt for some better items.

Lane looks at interception of missiles in the game *Missile Command*. He says that to guarantee an interception, it is not enough for our missiles to be faster than those of our enemies; they have to be faster by an amount that depends upon the enemy missile's location. After many derivations, Lane concludes that even for a game as simple as *Missile Command*, there is some really rich mathematics under it. Lane then looks at the games *Pac-Man* and *Ms. Pac-Man*. In games like those in this chapter, being able to anticipate the moves of adversaries is a significant benefit. Lane uses differential and integral calculus for deriving an equation for the path of a red shell.

Chapter 6 [Gaming complexity] starts with the game *Tetris* released for the Nintendo Entertainment System in 1989. In this game, the players manipulate shapes that mathematicians refer to as tetrominoes. Tetris is played in a rectangular grid. The tetrominoes fall into this grid, one piece at a time. The goal is to arrange the tetrominoes so that they fit nicely in the game board. To introduce complexity theory, Lane introduces the Bacon number which represents the shortest path between an actor and Kevin Bacon within a network of movies. Suppose we want to find the longest path between an actor and Kevin Bacon: It may seem difficult to find a solution, but it is easy to check the validity of a solution. The shortest path problem, which is concerned with finding the shortest path between two nodes in a graph belongs to a class of problems called **P**. Informally, the problems in **P** are the ones that computers can solve relatively quickly. The problem of finding the longest path between two nodes in a graph is quite unlike the shortest path problem, it is in a complexity class called **NP**. Informally, problems in **NP** are the ones whose solutions we can verify quickly. Researchers have shown that **Tetris is Hard, Even to Approximate**. So Lane jokes that playing a lot of Tetris may be the first step toward solving one of the most famous problems in mathematics (I would say in computer science). An added incentive is a million dollar prize available for anyone who resolves this problem. **NP** problems arise in many games not just *Tetris*. As an example, being able to identify a *Minesweeper* board as being consistent or not is **NP-complete**.

Lane looks at *Assassin's Creed: Revelations* in which an assassin has to determine the fastest route he can take to gather some fragments and return home. Surprisingly, this is an example of the celebrated mathematical problem known as the Traveling Salesman Problem. This problem is known to be **NP-hard**. Lane concludes by asking an interesting question:

Will someone solve one of the most famous open problems in mathematics by sitting on her couch and playing video games all day?

Lane answers that it would be safe to say no. He feels that playing these games probably would not make one a famous mathematician, but studying them more thoroughly simply might.

Chapter 7 [The friendship realm] looks at how video games have analyzed friendship and offers an understanding of the inherent mathematics. In the process, Lane examines several mathematical models for relationships. Arguably, some of these models are more complex than others, but additional complexity is not necessarily advantageous. After numerous derivations, Lane points out that the extent of mathematical sophistication arrived at is not really there in a game such as *The Sims*. He notes that friendship models in games such as *Far Cry 2* are clearly not an illustration of reality, and games such as *The Sims* are surely a bit closer to reality. The chapter investigates some classes of differential equations for modeling the dynamics of relationships. The models mostly highlight linear differential equations although some examples of nonlinear systems of differential equations are also discussed, although not in depth.

Chapter 8 [Order in chaos] digs into the mathematics of dynamical systems by exploring what makes a system chaotic. Lane looks at examples of chaotic systems inspired by games, and brings out the differences between a random system and a chaotic one. Lane looks at games with bouncing projectiles to present chaotic dynamics. We discover square configurations with projectiles having non-chaotic trajectories and we also come across elliptical configurations so projectiles have chaotic trajectories.

Chapter 9 [The value of games] looks at the real worth of video games in education. The common perception would be that we should be more focused on improving a student's ability in mathematics, rather than concentrate on mysterious examples of mathematics that arise if you look at some games from the right perspective. He gives three reasons why he believes that looking at games from a mathematical perspective, especially video games, is worthwhile. The reasons are

1. Games are relevant

   Lane argues that by showing students connections between mathematics and the things they spend so much of their time on (video games), we should be able to close the gap between what we want them to learn and what they find interesting.

2. Games have pedagogical value

   Lane mentions that many clever people are currently trying to figure out whether it is possible to use video games to engage students intensely in education. Lane makes an interesting comment:

   It's one thing to find interesting mathematics in games; it's another to use games effectively in the classroom in order to motivate the development of mathematics

   Lane says that statistics reveal that 97% of students are already playing games, so it would be unwise to overlook this medium.

3. Games have a precedent

   Video games are often viewed as being a waste of time. However, there is rich mathematical thinking to be harvested from games in general, not just those that are touted as being educational.

3 Opinion

Computer games/video games have been on the market for many years. They are attractive to people, especially young students, and they consume much of their precious time (a reason which drives others to eschew such games). The author has taken the unusual route of motivating those involved in teaching mathematics and to some extent physics to consider video games as a means of enticing students to learn those subjects. In order to fully appreciate the ideas in this book, the reader (perhaps a teacher of math or physics or anyone else) must have played some computer games and be familiar with some elementary concepts of math and physics. The book is certainly quite uncommon in its approach. The author stresses that games are finite by their very nature and they often become boring or obsolete once they are played lots of times. However, mathematics does not lose its charm as there are many new things to be discovered always.

Many students have a great aversion for math but few dislike computer/video games. Hence the author suggests that these students may be able to appreciate math better through computer games. Today, very few teachers would be kind enough to let their students play computer games since the teachers themselves would consider it as an utter waste of time. The truth is that a vast majority of teachers do not know the value of computer games for promoting sub-conscious learning. This book is an attempt by the author to make teachers revalue the beneficial aspects of computer games. It has some important lessons for video game developers too, for example, the need for realistic physics in games and some understanding of math so that the games are successful. The notes and the bibliography in the book will aid further reading. The book has many illustrations in color and is printed on durable acid-free paper.

Those who play video games may not realize that a game is basically a mathematical model of a virtual world simulated in real time on computers. So mathematics is ubiquitous in games. Many branches of mathematics including algebra, calculus, trigonometry, statistics etc. have a major role in video games. AI, physics-based modeling, path-finding algorithms, and collision detection are important for modern video games, of course in addition to programming. Teachers, enthusiasts of computer games, game developers, those who love math or physics, laypersons, students, and even those who hate math or physics but love computer games will benefit by reading this book. In this book there is plenty of computer science (thanks to discussions about algorithms and complexity theory) and even physics, not just mathematics, so I feel a more appropriate title for this lucid book might be:

Power Up: Unlocking the hidden mathematics, computer science, and physics in video games

Review of[5]
Probability and Computing:
Randomization and Probabilistic Techniques
in Algorithms and Data Analysis (Second Ed.)
by Michael Mitzenmacher and Eli Upfal
Cambridge University Press, 2017
467 pages, Hardback, $69.99

Review by
Aravind Srinivasan, `srin@cs.umd.edu`
Dept. of Computer Science and UMIACS
University of Maryland, College Park, MD 20742, USA

1 Introduction

The power of randomization in the computational context is one of the fundamental discoveries of computer science; furthermore, statistical and probabilistic reasoning is indispensable for modern-day data analysis and data science. In this significantly-expanded second edition, the authors (continuing to assume exposure to just basic discrete probability) present a unified treatment of probabilistic concepts useful for modern computer science and data science.

2 Summary

Chapters 1 and 2 are devoted to a recap of basic probability and some of the very elegant, easy-to-state/analyze randomized algorithms. A nice illustration of where this book differs from the other excellent books on randomization comes as early as in Chapter 1, when the book introduces Bayesian reasoning with two applications. The first is to a slightly non-rigorous (but practically useful) approach to constructing a simple Bayesian classifier for, e.g., spam filtering. The second is on an interesting question: suppose we have a decision problem, say in RP, and run many iterations of the corresponding randomized algorithm – how will a Bayesian update their belief of what the answer is, as a function of the number of iterations? As a concrete example, consider Freivalds' verification of matrix multiplication, where, given matrices A, B and C, we want to verify very quickly if $AB = C$. Our prior probability of the truth of this could be, say, $1/2$: as successive iterations give more and more evidence for this fact (through the well-known Freivalds' analysis), how should we, as a Bayesian, increase our belief in this fact? I found these two applications so compelling that I taught them in a class at the earliest opportunity. Given that Bayesian reasoning will play an increasingly-major role in the future (say, for AI agents), I see this as a very appealing way to introduce the concept.

Chapters 3 and 4 are on moment bounds and applications, and the treatment is largely what we typically find in textbooks. One of the interesting topics covered here is how permutation routing on the butterfly (say, for routing random permutations) shows a fundamental difference from the easier case of the hypercube: conditional on the path for a particular packet, the expected number of other packets that share an edge with this path is $\Omega(n^2)$, leading to an introduction to delay-sequence arguments.

[5]©2018, Aravind Srinivasan

Chapter 5 covers basic balls-and-bins (with the more-sophisticated "power of k choices" studied in Chapter 17), basic hashing such as Bloom filters (again, with the more-involved Cuckoo hashing covered in Chapter 17), and basic random-graph models. In a taste of the emphasis in this book of understanding the limits of various distributions, the Poisson approximation is also introduced in this chapter.

Chapter 6 presents a basic coverage of the probabilistic method, including the Lovász Local Lemma; Chapter 7 is an introduction to random walks and Markov Chains, with coupling covered in a later dedicated chapter, Chapter 12.

The book departs from several sister texts on randomness and computation, starting with Chapter 8. For instance, Chapters 8 and 9 cover continuous distributions, Poisson processes, continuous-time Markov processes, basic queuing theory, the normal distribution and the Central Limit Theorem, the multi-variate normal, maximum-likelihood estimation, the EM algorithm etc. *I believe the power of this book lies here, by no longer viewing randomized algorithms as primarily a "theoretical computer science" or "discrete mathematics" topic, but also by inviting anyone who wishes to learn the probabilistic/statistical foundations of data science, for instance.*

Chapter 10 is on basic information theory, and Chapter 11 studies MCMC and careful Monte-Carlo sampling (e.g., in DNF counting). Next, as mentioned above, coupling (a powerful technique to bound the mixing time) is presented in Chapter 12. Chapter 13 is a thorough introduction to Martingales, and to simple but very useful tools such as Wald's Equation that are perhaps not as well-known as they should be.

Chapter 14, on aspects of machine learning, is one of the newer topics in this second edition. It covers basic notions including PAC learning, learnability as a function of the VC-dimension, Rademacher complexity, etc. This is a welcome chapter as the concepts are quite simple and these (and related) concepts are increasingly important.

Chapter 15 is on another topic related to hashing: how k-wise independence offers a rigorous approach to constructing usable hash functions with provable properties (as opposed to the idealized models that are often used in analyzing, say, Bloom Filters). Chapter 16 is on an aspect of many human-engineered systems: power laws ("long tails" in the popular press) – their sources, analyses, etc. Chapter 17 concludes with balanced allocations, the power of k choices, and Cuckoo hashing.

3 Opinion

By assuming just an elementary introduction to discrete probability and some mathematical maturity, this book does an excellent job of introducing a great variety of topics to the reader. I especially liked the coverage of the Poisson, exponential, and (multi-variate) normal distributions and how they arise naturally, machine learning, Bayesian reasoning, Cuckoo hashing etc. There is a broad range of exercises, including helpful ones on programming to get a feel for the numerics: e.g., Section 11.6 is an assignment to the reader on minimum spanning trees in the complete graph with random edge-weights in $[0, 1]$, including a discussion on memory issues and re-seeding the random-number generator! This connection to practice is unusual and very commendable.

As is inevitable in any work, there are some shortcomings in the book, although they are few and relatively minor. For example, more extensive references would be useful: at least major ones that are used such as Moser-Tardos. I understand the preference to have a limited list of references, but major discoveries such as the Moser-Tardos algorithm do appear to need a reference. It would be useful to define RP, coRP, BPP and ZPP. The (perhaps surprising) current consensus opinion that BPP = P can then be presented, and footnote 3 in page 175 (on randomized algorithms for 3-SAT) could say more concretely that "This would imply NP = ZPP, which would be far-reaching."

Overall, I would highly recommend this book to anyone interested in probabilistic and statistical foundations as applied to computer science, data science, etc. It can be taught at the senior undergraduate or graduate level to students in computer science, electrical engineering, operations research, mathematics, and other such disciplines. I reiterate what I said above: I believe the power of this book lies in regarding randomized algorithms as more than a "theoretical computer science" or "discrete mathematics" topic, also providing the probabilistic/statistical foundations of data science. It was a pleasure to read this book, and especially to enjoy its unusual features mentioned in this review.