

available at www.sciencedirect.comjournal homepage: www.elsevier.com/locate/cosrev

Book review

Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani. *Algorithms*. McGraw Hill, Boston (2007). x+320 pp., Paperback \$33.75, ISBN: 978-007352340-8

Jon Kleinberg, Éva Tardos. *Algorithm Design*. Pearson/Addison Wesley, Boston (2006). xxiii+838 pp., Hardcover \$103, ISBN: 978-032129535-4

1. Starting point

The design and analysis of algorithms is a cornerstone of computer science curricula. Accordingly, there are several textbooks around that cover the subject. Most of them, notably the volumes by Sedgewick and by Cormen, Leiserson, Rivest, and Stein will include all the elementary algorithms and data structures. In this review, we take a look at two recent additions to the set of textbooks, which start at the advanced undergraduate level: “Algorithm Design” by Kleinberg and Tardos (both with Cornell), which will be cited as [KT], and “Algorithms” by Dasgupta (UC San Diego), Papadimitriou (Berkeley), and U. Vazirani (Berkeley), cited as [DPV].

2. Approaches

Both books are primarily targeted at undergraduate students and are meant as texts for the undergraduate course on algorithms. They presuppose that the first courses on programming and elementary algorithms and data structures have been completed. The difference: [DPV] stays in a range of topics suitable for undergraduates, while the material of [KT] extends way beyond that, so that this book can also be used for an introductory graduate course, or for further self-study.

In [DPV] computational problems are proposed and the algorithms are developed in a traditional style: The computational problems are already there in their distilled form, and need to be solved, sometimes applications or motivating examples are given. In correctness proofs and running time analyses the authors emphasize the central mathematical ideas and arguments rather than tedious technical details. The book avoids spelling out lengthy

arguments in full detail, and either refers to “observations” by means of examples or leaves some of the detailed calculations to the reader. In that, a certain maturity is expected on the part of the reader that enables him/her to meet the challenge to fill in missing pieces or details. For readers that possess this ability this book is pleasurable to work with as it is packed full of ideas.

In contrast, the authors of [KT] try a novel approach of leading the way to algorithms. Each and every algorithmic problem in the book is discussed along a fixed pattern. A problem is introduced starting from a description from the “real world” (“the problem”), so that the first step to be done is to isolate the neat, structurally clean computational problem hidden in the real world description (“formulating the problem”). The next step (“designing the algorithm”) carefully leads the reader through the steps of the construction of the algorithm, as it might have been when it was first found, including sometimes the discussion of wrong approaches. Finally (“analyzing the algorithm”), correctness—or optimality, or approximation ratio, or whatever is claimed about the algorithm—and runtime analysis follow, in sufficient detail to be followed easily.

Both books offer a wealth of exercises. In [DPV], we find more than 250 exercises (on a total of about 70 pages), mostly also in a traditional style. The style of the more than 200 exercises in [KT] (on a total of about 135 pages) is in keeping with the overall approach of the main text: situations are described that ask for an algorithmic solution, but the whole design path has to be walked through.

3. Topics

3.1. Prerequisites

Both books assume the reader has the knowledge and ability gained from having completed the basic CS courses, in particular the use of basic data structures and basic algorithms for sorting and searching. In [DPV], this is just assumed, and some material that will occur in these courses (like priority queue implementations via binary heaps or quicksort) is covered in the exercises. Some preliminaries

are reviewed in Chapter 0 (the “Prologue”, 8 pages), which introduces algorithms, polynomial and exponential running times, and the $O(\Omega, \Theta)$ notation. All graph traversal algorithms are developed from scratch later in the book.

In [KT], Chapters 2 and 3 serve to review material that students are expected to know from their first CS courses (but might not, due to differences in the organization of the courses). Chapter 2 (36 pages) deals with the basics of algorithm analysis. It discusses different growth rates for running times of algorithms, the special role played by polynomial running times, the $O(\Omega, \Theta)$ notation, and examples of algorithms with running times of different growth rates. As an elaborate example, the implementation of priority queues as binary heaps is described and analyzed.

In Chapter 3 of [DPV] (15 pages), graphs and digraphs and the basic methods for exploring and decomposing them are discussed. Chapter 3 of [KT] (31 pages), explicitly marked as a review chapter, is devoted to the same topic. Thus, in both books we find depth-first search, breadth-first-search and connected components in undirected graphs, for directed graphs depth-first search, classification of edges, acyclicity test and topological sorting. In [DPV], Kosaraju’s algorithm (although the name is not used) for strongly connected components is described and analyzed, while in [KT] the existence of a linear-time algorithm for this problem is only stated.

3.2. Starting out

Chapter 1 of [DPV] starts the treatment of algorithms in an unconventional way by turning to a classical topic first, namely to algorithms for numbers: binary arithmetic, modular arithmetic, in particular fast modular exponentiation, and the Euclidean Algorithm. Primality testing is presented in terms of the Fermat test—the problem that the Fermat test will fail for the (very rare) Carmichael numbers is treated briefly and picked up in the exercises. In passing, the reader is introduced to the idea of a randomized algorithm and is reminded of the elements of group theory. Next are algorithms that use numbers (integers) and their properties: Armed with a primality test and reminded of the prime number theorem on the density of primes, we can efficiently generate random primes; given large prime numbers, we can do cryptography, in particular can implement the RSA scheme. Modular arithmetic with a prime modulus can also be used to construct universal hash classes, which are useful for storing sets. In this first section of [DPV], the three basic questions one should ask about proposed algorithms are hammered into the readers’ minds (at least so one hopes): Given an algorithm: (a) Is it correct? (b) How long does it take? (c) Can we do better? All this material is covered in 27 pages, which demonstrates a very interesting feature of [DPV]: The authors aim at covering a lot of material. They focus on a clear explanation of how the algorithm under consideration works (and sometimes describe ways of thinking that might have lead the way to finding it); in the correctness analysis they concentrate on the “crisp mathematical idea that makes the algorithm work”. This means that they avoid going into the tedious details of more down-to-the-earth calculations or induction proofs.

The introductory chapter of [KT] (Chapter 1, 18 pages) approaches the topic “algorithm design” at a much more leisurely pace. As a first example, the stable matching problem is described, and the Gale–Shapley algorithm is developed. Next, five further “representative problems” are described that represent different possibilities for strategies and qualities of algorithmic solutions. As mentioned before, the reader then is invited in Chapters 2 and 3 to recall material about algorithm analysis and about basic graph exploration techniques.

3.3. Divide-and-conquer algorithms

The D–a–C algorithm design paradigm is treated in separate chapters in both books: Chapter 2 in [DPV] (26 pages) and Chapter 5 in [KT] (34 pages). In [DPV], the strategy is presented by means of a series of familiar examples: The Karatsuba–Ofman method for integer multiplication, Mergesort, the randomized algorithms for median and selection, Strassen’s matrix multiplication algorithm. The master theorem for solving D–a–C recurrences is stated, and proved in its simplest form. Finally, as a very elaborate application of D–a–C, the Fast Fourier Transform algorithm is presented. Some other “usual suspects” for the D–a–C paradigm like Quicksort are only briefly mentioned in the text, the details of the partition procedure and the analysis are proposed as exercises. In [KT], the D–a–C strategy is introduced by means of the mergesort algorithm, solution strategies for recurrences are discussed for several special situations (splitting into 2 subproblems, the master theorem is missing). A geometric problem, closest pairs in the plane, provides the next example, before turning back to the more conventional topics integer multiplication (by Karatsuba–Ofman) and Fast Fourier Transform, which is treated in full.

3.4. Paths in networks

Chapter 4 of [DPV] (17 pages) considers path problems in graphs, namely: simple edge-counting distance by breadth-first search, Dijkstra’s algorithm (with two slightly different derivations), some remarks on implementations of priority queues and a runtime analysis based on the properties of these data structures. The section concludes with the Bellman–Ford algorithm for digraphs with negative cycles and the special case of shortest paths in directed acyclic graphs. The exercises include the implementation of binary heaps in arrays, with analysis.

3.5. Greedy algorithms

The greedy algorithm design paradigm is the topic of separate chapters in both books: Chapter 5 in [DPV] (21 pages) and Chapter 5 in [KT] (68 pages).

The [DPV] chapter starts out with Kruskal’s algorithm for minimum spanning trees. Not greedy, but needed in Kruskal’s algorithm: the union-find data structure, fully-fledged with path compression and a compact proof of the amortized time bound $O(\log^* n)$ for find operations. Prim’s (or Jarník’s) minimum spanning tree algorithm follows. The

construction of Huffman codes is next, which provides the authors with an opportunity to tell the reader a little bit about entropy. The greedy algorithm for the satisfiability of sets of Horn clauses is described. Finally, set cover—yes, the approximation algorithm with approximation ratio $\ln n$. (The framework for approximation algorithms is discussed only later, in Section 9.2.) The corresponding chapter in [KT] introduces the greedy strategy and analysis methods by means of scheduling problems (interval scheduling, Min-lateness scheduling), introduces and analyzes the optimal caching strategy “farthest in the future”, and only then turns to a comprehensive discussion of strategies for the MST problem (“cut property” and “cycle property”) and algorithms (Prim, Kruskal, Reverse-Delete), including a union-find structure (without path compression). Related to Kruskal’s algorithm is a solution to the Maximum spacing clustering problem. Huffman codes and data compression are discussed in full. The chapter also features Dijkstra’s algorithm and finishes with an algorithm for Minimum-cost-arcborescences as a quite complex application of the greedy strategy.

3.6. Dynamic programming

Of course, both books treat dynamic programming as a fundamental design strategy and each of them devotes a chapter to the topic: Chapter 6 in [DPV] (21 pages) and Chapter 6 in [KT] (56 pages). In [DPV], the principle is explained by some standard examples: longest subsequence, edit distance, knapsack, chain matrix multiplication, all-pairs shortest paths, and the TSP (leading to the $O(n^2 2^n)$ -time algorithm). The abstract way of viewing a problem with its subproblems as a directed acyclic graph (dag) is described, as is memoization as a means of reconciling recursion with the dynamic programming approach. In [KT], the DP paradigm is introduced for the weighted interval scheduling problem, the treatment already including recursive formulation and memoization. After discussing the principles, another nonstandard problem is next: segmented least squares approximation. The knapsack problem is treated. Alignment problems in different versions are a central issue in this chapter of [KP] (“RNA secondary structure”, edit distance, including the divide-and-conquer construction in linear space), as are shortest path problems with negative edge weights (Bellman–Ford-Algorithm and “push-based” variants) and negative cycle detection.

3.7. Network flow

It can definitely be said that algorithms for network flow problems are of foremost interest in [KT]. All of Chapter 7 (75 pages) is devoted to this area. Topics treated include the Max-Flow problem (Ford–Fulkerson algorithm, Max-Flow-Min-Cut-Theorem, integrality). The runtime bound for the algorithm is improved by a scaling method; the preflow-push method with its strongly polynomial running time, is discussed in full. In the remainder of the chapter, various variants and applications of the network flow problem are investigated: more common ones like bipartite matching, edge-disjoint paths, and circulations with demands, but also “airline

scheduling”, “image segmentation”, and the very entertaining “baseball elimination problem”. In contrast, [DPV] integrates the discussion of the basic network flow algorithm and the Max-Flow-Min-Cut-Theorem into the chapter on Linear Programming; only in the exercises are applications beyond bipartite matching discussed.

3.8. Linear programming

Chapter 7 of [DPV] (35 pages) is devoted to what the authors of [DPV] consider the other “sledgehammer of the algorithms craft” (besides dynamic programming): linear programming. The treatment of this problem type is much more extensive here than one would expect in algorithms texts. After a gentle introduction by geometric examples, the simplex algorithm is announced as a method for solving LP instances. LP is described as a universal tool to formulate optimization problems in a way so that a general package, your favorite LP-solver, can be used to solve them. The concept of reduction among problems is introduced as a method for solving problems (here: by reducing optimization problems to LP). As a prime example, the Max-Flow problem is demonstrated to reduce to LP, and the Ford–Fulkerson algorithm is introduced as just a special way of applying the simplex algorithm. Duality is explained in a concise way, and the Duality theorem is stated and declared to be a consequence of properties of the simplex algorithm. Zero-sum games are introduced and their connection to LP and duality is demonstrated. The simplex algorithm is explained in a mathematically clean manner, but not quite ready for being transferred into code. Running time issues are discussed and the ellipsoid method and interior point methods are mentioned. Finally, it is shown that boolean circuit evaluation reduces to integer linear programming, thus hinting at the inherent difficulty of the latter problem.

3.9. NP-completeness

Both books offer one chapter on the theory of NP-completeness and on problems that are intractable because of being NP-hard: Chapter 8 in [DPV] (31 pages) and Chapter 8 in [KT] (50 pages). In [DPV], the discussion of NP-completeness is based on the problem type “search problem” (“given x , find a satisfying solution if one exists, otherwise report that there is none”), that means that also P and NP are described as classes of search problems. In [KT] the more conventional type “decision problem” is used, but of course NP is introduced as the class of problems with efficiently checkable certificates. In both books, more or less the standard series of NP-complete problems is introduced and proved NP-hard by reductions respectively reduction sequences from 3-SAT. (In [DPV], graph coloring problems are missing, but integer linear programming and zero-one-LP are included. That Hamiltonian cycles are called “Rudrata cycles” in [DPV] is amusing and confusing—if the authors were consistent in such matters, Prim’s algorithm would have to be renamed “Jarník’s algorithm”. The use of the notation “ \rightarrow ” instead of the standard “ \leq_p ” is confusing for those readers who also refer to other books.) For the NP-completeness of SAT both books give rather informal arguments—not quite satisfying

for readers who look for a fully rigorous argument, but obviously anything else would lead far off the track in algorithms texts.

3.10. Other hardness results

In [KT], the short Chapter 9 (16 pages) is devoted to the discussion of problems solvable in polynomial space, in particular certain games—the standard Quantified Boolean Formulas problem can be regarded as the question for a winning strategy in a general game—and planning problems, and gives some PSPACE-completeness proofs, thus widening the view for intractability beyond NP and co-NP. In [DPV], at the end of the NP-chapter, a brief discussion of undecidability can be found (with an informal proof of the undecidability of the halting problem).

3.11. Coping with hard problems, briefly

In [DPV], the situation that even NP-hard problems must be solved is discussed in Chapter 9 (22 pages), which offers a tour of different strategies. This includes backtracking and branch-and-bound, approximation algorithms (the problems vertex cover, k -clustering, Δ -TSP are considered, as is the approximation scheme for Knapsack). Local search heuristics are discussed for the examples TSP and graph partitioning (a relaxed version of Balanced Cut). Finally, simulated annealing is briefly discussed as a means for dealing with local optima.

3.12. Coping with hard problems, elaborate

In [KT], three full chapters, complete with exercises, are devoted to strategies for dealing with hard problems. These chapters partly discuss basic strategies, applied to the standard problems, but then also dive deeply into advanced and more challenging material.

Chapter 10 (38 pages) is devoted to fixed-parameter algorithms, which have polynomial running time if one parameter in the problem is fixed. Examples considered are finding small vertex covers and circular arc coloring with few colors. Next, it is demonstrated that some hard problems like independent set become easy on trees. The second half of the chapter offers a very nice introduction to fixed-parameter algorithms (on the basis of dynamic programming) for graphs with bounded treewidth—definitely an advanced, but important topic.

Chapter 11 of [KT] (50 pages) studies approximation algorithms for NP-hard optimization problems. The set of problems considered include Min-makespan, k -center, set cover, vertex cover without and with weights (an opportunity to introduce the pricing method), and disjoint paths. Further, the chapter introduces linear programming and the existence of polynomial-time algorithms for LP as a central tool in approximation algorithms for problems like vertex cover or load balancing. The approximation scheme for the knapsack problem is presented as well.

Chapter 12 of [KT] (40 pages) discusses several aspects of local search heuristics. The general principles of local search, of the Metropolis algorithm and of Simulated Annealing are introduced, and the local search strategy for finding a

stable configuration in Hopfield neural networks is analyzed. More complicated examples include an image segmentation problem. Finally, the opportunity is taken to introduce best-response dynamics in games and Nash equilibria, and discuss the relationship to local search strategies, thus introducing the reader to the area of algorithmic game theory, an active research field.

The final Chapter 13 of [KT] (70 pages) is a complete introduction into the field of randomized algorithms. (The topic of randomized algorithms forms a “virtual chapter” in [DPV], with half a dozen examples scattered across the book.) The treatment in [KT] is quite systematic, even including a section devoted to basics from probability. Problems and algorithms treated here include contention resolution (symmetry breaking), Karger’s Minimum Cut algorithm, the simple approximation algorithm for MAX-3-SAT, median finding and Quicksort (in a somewhat strange version tuned and slowed down so as to make the analysis simpler), universal hashing, and an extended treatment of a closest pair algorithm by Golin et al. Finally, the reader is led back to the caching problem encountered already in the chapter on greedy algorithms, and is introduced to the randomized marking algorithm that offers a competitive ratio of $O(\log k)$. The analysis of a packet routing protocol (using Chernoff bounds) is the last algorithm analyzed in this section. The final “Epilogue” offers a glance at a totally different type of algorithmic problem: algorithms that run forever, discussed by means of the example of operating a routing switch.

3.13. Quantum algorithms

The concluding Chapter 10 of [DPV] (18 pages) offers a look into the (quantum computing) future. The discussion in this chapter explains the necessary fundamental notions and leads up to the quantum algorithms for Fourier Transform and for factoring—not in full detail, but at a level that makes it possible to grasp the central ideas.

4. Special aspects

4.1. Indexing, references, notes

[DPV] includes 2 pages of subject index. This is too short. For example, the term “Ford–Fulkerson” appears in the text, but not in the index, similarly for “maximum flow” or “randomized algorithm”. Also, there are only two pages of “Historical notes and further reading” with seven references overall. This parsimony, of course, is in accordance with the main purpose of the book as an undergraduate text, but it is far from an ideal solution. [KT] offers one or two pages of “Notes and further reading” with each chapter, $9\frac{1}{2}$ pages of references to the literature, and a very carefully compiled subject and name index, taking up 24 pages. These features make it easy to find one’s way through the book and into the research literature.

4.2. Special features

[DPV] includes a series of “boxes”, in which material is highlighted in a compact form that does not belong to the main thread. These boxes provide little historical stories (e. g. about the slow spread of the FFT algorithm), glimpses at more advanced material (different heap types), or remarks that point to more abstract underlying mathematical structures. Also, [DPV] features pictures of some famous mathematicians that have contributed to the area of algorithms: Fibonacci, Euclid, Gauss. This looks friendly and inviting, but it is not carried through very systematically.

4.3. Exercises and problems

Both books offer many exercises. However, as mentioned before, the type of exercises is very different.

In [DPV], the exercises range from the routine application of algorithms through extensions and variants, or filling in proofs, to quite challenging extensions or the analysis of algorithms and data structures not described in the text (e. g., priority queue operations or quicksort or Tarjan’s algorithm for biconnected components). The degree of difficulty of the exercises is vastly different, with no indication of what is easy and what is tough.

The style of the exercises in [KT] (on a total of about 135 pages) is in keeping with the overall approach of the main text: situations are described that ask for an algorithmic solution, but the whole design path has to be walked through: identify and formulate the computational problem in the situation, employ known algorithms and design strategies to find an algorithm tailored to the problem, prove correctness, analyze running time. Of course, also here the level of difficulty of the exercises ranges from easy to challenging. A special and very useful feature of [KT] are two or three “solved exercises” given at the end of each chapter, which serve as examples for how the exercises may be approached.

4.4. Pseudocode

In both books, in many cases, algorithms are developed to a level of precision where they can be described in pseudocode.

In [DPV], this feature is attended to more strictly than in [KT]. (Exceptions are e.g. the simplex algorithm and the bipartite matching algorithm, which have no pseudocode version in [DPV].) Of course, in pseudocode sections some actions are described in words, and set and function notation is used freely. Still, the way to an actual implementation in a programming language is made shorter and less error-prone in this way.

In [KT], pseudocode is also given for quite a few algorithms, but not as systematically as in [DPV]. This may in some cases make it more difficult to derive an actual program from the description of the algorithm. For example, none of the MST algorithms is given in pseudocode, nor are the methods for the union-find data structure. For the Huffman code algorithm the actual method one will implement is much simpler than indicated by the pseudocode formulation. Also, the syntax in the pseudocode segments does not seem to follow a well-defined system.

In both books, assignments are written with the equality sign—easy to read for C, C++, and Java programmers, not so nice in general, in particular if comparisons are also written as (in)equalities (“if $y = 0$ ”). If Pascal’s “:=” looks old-fashioned, why not use “←” systematically?

5. Conclusions

5.1. How easy or hard is the text to work with?

Although the pages of [DPV] are packed with information, the reader never has the feeling that the authors are short of space: the tone always is friendly and easygoing. Only after reading 20 pages one realizes how much material has been covered. This style is good for the gifted student who wants to see the essential methods for the design and analysis of algorithms. Annoying details, routine calculations, tedious correctness proofs by induction of an algorithm whose underlying idea has been discussed before are left out or given as exercises. For example, the algorithm for finding a Huffman code is motivated in the text, but no correctness proof is given—which means the reader will not learn from the book how to do such a proof. Weaker students who need to be told such simple things repeatedly before becoming fluent enough to produce them themselves may be at a loss here. Even very good students will not learn from the book (but hopefully from the accompanying problem sessions) how to write complete proofs themselves, down to all the details, an ability one will need when writing research papers.

The text of [KT] is very well and elegantly written and a pleasure to read. Readers who just want to learn algorithms for already well-carved problems will find the first part of the problem discussions too lengthy, but for the serious student of algorithmics this book adds a quality I haven’t seen in other books before: the experience that in real life the problems to be solved do not pop up as condensed 10-line paragraphs that already include the description of the needed data structures, but usually are vague and leave open the task of problem analysis and structuring. I think the text trains the student very nicely for coping with this situation. The feature that the analysis of most algorithms is carried out in detail gives a lot of training material for writing proofs and makes the book quite suitable for self-study.

5.2. Which type of reader are these books written for?

Students whose aim it is to learn a lot about the inner workings of a lot of algorithms, about the essential design strategies and the central arguments used in the analysis of algorithms, who are bright and have a solid basic training in arguing about algorithms will get along well with [DPV]. This book also contains more material than one can fully cover in a 1-semester course, so there is room left for self-study. The exercises, apart from practice in applying the techniques, also further extend the technical material. According to its central philosophy the main text of the book spares the reader technical details and lengthy formal proofs. Even those students who are inclined toward theoretical research will partly profit from this feature of the book because they are forced to work

out many of the details for themselves—a very good training. It is not a book to learn such techniques from, so doing self-study with this book is possible, but may be difficult for readers who are not already well-versed in doing proofs.

The reader of [KT] can be one of several varieties. It is a great textbook for the undergraduate student who will see not only computational problems and algorithms but also the way from sometimes imprecise problem descriptions to the structural core. The reader is led to the algorithm in a careful manner, sometimes indicating approaches that turn out to be blind alleys, so that the process of finding the algorithm can be duplicated by the student. In most cases, the analysis is carried out in full, so that even this part is not hard to work through. The text reaches way beyond any undergraduate course in its extensive treatment of algorithms for the maxflow problem, its coverage of approximation algorithms, search heuristics, and randomized algorithms. If proofs are given, they are given in full, even spelling out simple observations or arguments, so on the whole it will be easier to digest than [DPV], and it is quite amenable for self-study.

5.3. Which book would I recommend?

This partly is a question of your budget: you get three copies of [DPV] for one of [KT]. Regarding the pure technical content, in the [DPV] book the material is packed more densely. In this way, regarding techniques, one will get more than one half of the material covered in [KT]. This wealth of material, the integrated treatment of linear programming, the chapter on quantum algorithms, and the exercises make the [DPV] book interesting, especially for readers who are ready to digest a lot of technical material fast. Any student who has mastered this material will be very well trained in algorithmics as it is traditionally understood—but it will be rough going in places.

[KT] is attractive in its novel approach to algorithmics, including the path from the real world to the problem formulation. Some students who can't wait might find that they have to read a lot of text to get to the technical core of an algorithmic problem. But I feel this approach brings in a new style, looking at the field in a way that goes beyond the techniques alone. This setup adds new appeal to the study of algorithms. In the student, this approach stimulates a broader way of thinking about algorithms and the ways they are applied than in earlier decades, and it trains him/her for situations he/she will find themselves later, when confronted with real life algorithmic problems. The selection of problems, of algorithmic techniques and of design patterns covers a vast range. Students who have worked through this material will be well trained in many algorithmic techniques as well as in spotting problem patterns in real world situations. In addition, the writing style is excellent, making reading a pleasure. So, yes, especially for students interested in the field of algorithm design beyond the techniques alone: this is a great book.

Both books serve their purpose very well, but the aims and the styles are different. I personally find both of them appealing and I think a student or a teacher will profit a lot from either book or from both of them in combination. So, if you can afford it: get both books. Anyway, I have both of them on my shelf, ready to hand.

Martin Dietzfelbinger
Faculty of Computer Science and Automation,
Technische Universität Ilmenau,
D-98684 Ilmenau,
Germany

E-mail address: martin.dietzfelbinger@tu-ilmenau.de.