The Book Review Column[1]

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

In this column we review the following books.

1. **Fair Division and Collective Welfare** by Hervé Moulin. Reviewed by Varsha Dani. How do you divide up a resource fairly? What is 'resource'? What is 'fair'? What is 'divide'? This book rigourous asks questions along these lines and proposes answers.

2. **Algorithms: Sequential, Parallel, and Distributed** by Kenneth A. Berman and Jerome L. Paul. Reviewed by William Schmeister. An undergrad textbook on algorithms, but with some additional topics as well.

3. **Algebraic Complexity Theory** by Peter Bürgisser, Michael Clausen, and M. Amiu Shokrollahi. Reviewed by Anthony Widjaja. Algebraic Complexity Theory studies the complexity of a problem by the number of operations (e.g., additions, multiplications) are needed to solve it. This is a graduate level textbook on this subject.

Books I want Reviewed

If you want a FREE copy of one of these books in exchange for a review, then email me at gasarchcs.umd.edu

Reviews need to be in LaTeX, LaTeX2e, or Plaintext.

Books on Algorithms and Data Structures

1. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms* by Hromkovic.

2. *Combinatorial Optimization: Packing and Covering* by Cornuejols.

3. *An Introduction to Data Structures and Algorithms* by Storer.

4. *Handbook of Algorithms for Wireless Networking and Mobile Computing* edited by Boukerche.

5. *Handbook of Bioinspired Algorithms and Applications* edited by Olariu and Zomaya.

6. *Solving Polynomial Equation Systems II: Macaulay's Paradigm and Grobner Technology* by Mora.

Books on Cryptography and Security

1. *Complexity Theory and Crytography: An Introduction to Cryptocomplexity* by Rothe.

2. *Complexity and Cryptography: An Introduction* by Talbot and Welsh.

3. *Elliptic Curves: Number Theory and Cryptography* by Larry Washington.

4. *Crytographic Applications of Analytic Number Theory: Complexity Lower Bounds and Pseudorandomness* by Shparlinski.

[1] © William Gasarch, 2006.

5. *Cryptography and Computational Number Theory* edited by Lam, Shparlinski, Wang, Xing.
6. *Coding, Cryptography, and Combinatorics* edited by Feng, Niederreiter, Xing.
7. *Foundations of Computer Security* by Salomon.
8. *Coding for Data and Computer Communication* by Salomon.
9. *Foundations of Logic and Mathematics: Applications to Computer Science and Cryptography* by Nievergelt.

Books on Coding Theory

1. *Introduction to Coding Theory* by van Lint.
2. *Error Correction Coding: Mathematical Methods and Algorithms* by Moon.
3. *Block Error-Correcting Codes: A Computational Primer* by Xambo-Descamps.
4. *Coding Theory: A First Course* by Ling and Xing.
5. *Algebraic Coding Theory and Information Theory: DIMACS Workshop* Edited by Ashikhmin and Barg.
6. *Authentication Codes and Combinatorial Designs* by Pei

Game Theory

1. *The Game's afoot! Game Theory in Myth and Paradox* by Alexander Mehlman.
2. *An Introduction to Game-Theoretic Modelling* by Mestertson-Gibbons.

Combinatorics Books

1. *A Course in Combinatorics* by van Lint and Wilson.
2. *Weighting the Odds: A Course in Probability and Statistics* by Williams.
3. *A Beginners Guide to Graph Theory* by Wallis.
4. *Graphs and Discovery: A DIMACS Workshop* Edited by Fajilowicz, Fowler, Hansen, Janowitz, Roberts. (About using software to find conjectures in graph theory.)
5. *Combinatorial Designs: Constructions and Analysis* by Stinson.
6. *Ordered Sets: An Introduction* by Schroder.

Logic Books

1. *Logicism Renewed: Logical Foundations for Mathematics and Computer Science* by Gillmore.
2. *Essay on Constructive Mathematics* by Edwards.

Misc Books

1. *Domain Decomposition Methods– Algorithms and Theory* by Toseli and Widlund.
2. *The Elements of a Computing Systems: Building a Modern Computer from First Principles* by Nisan and Schocken.
3. *Handbook of Computational Methods of Integration* by Kyther and Schaferkotter.
4. *Polynomials* by Prasolov.
5. *An Introduction to Difference Equations* by Elaydi.
6. *Difference Equations: From Rabbits to Chaos* by Cull, Flahive, Robson.

Review[2] of
Fair Division and Collective Welfare
Author: Hervé Moulin
Publisher: MIT Press, 2003
$40 Hardcover, $22 Paperback

Reviewer: Varsha Dani[3]

The founders (or rulers) of a society are faced with a number of questions of social welfare. Here are some possibilities:

- Should everyone have an equal say in the making of rules?
- Should the rich be taxed more than the poor?
- Who should bear the expense of things that benefit everyone, such as building of roads?
- Should minorities receive special treatment?
- Which city gets to be the capital?

Such questions have been answered quite differently by different societies and cultures in different times. In dictatorial regimes they may have been decided arbitrarily, without reference to the well-being of the people. Ideally, the rules should be "fair" to all the individuals to whom they apply. This book introduces the reader to some philosophical, economic and game theoretic interpretations of "fairness" and "social welfare."

In the next few sections, we attempt to give a quick taste of some of the broad topics considered in the text.

Utilitarian social welfare

The basic principle underlying fairness is that equals should be treated equally. What about when the agents in the game are not equals? There are a number of principles governing our notion of fairness in this case. *Compensation* is the idea that an underprivileged agent may be allocated a larger share of the common resources in order to equalize their utility in the end result. Affirmative action is an example of this. *Reward* refers to an unequal allocation that is justified by unequal input. Investors with a greater risk should get a larger share of the profits. *Fitness* is the idea

[2]© 2006, Varsha Dani

[3]Department of Computer Science, University of Chicago, Chicago, IL 60637, email: `varsha@cs.uchicago.edu`

that resources should be allocated to the agents that will make the best use of them. Thus in a situation where medical attention is in short supply, those with the highest chance of recovery are treated preferentially.

A very important notion for a resource allocation problem is that of Pareto optimality. An outcome or allocation x is Pareto inferior to another outcome y if a proposed change from x to y (by an agent who strictly benefits from the swap) meets with no opposition from the other agents (i.e., nobody is worse off than before in the new outcome.) An outcome is said to be Pareto optimal if it is not Pareto inferior to any other. Pareto optimality is desirable because it is efficient: from such an outcome it isn't possible to improve anyone's condition without hurting someone else.

Consider the problem of the location of a desirable facility such as a fire station. Nobody wants it to be too far from them, since proximity to the station increases the chances of being saved in the event of a fire. Thus each person in the town has a utility function that measures their satisfaction with the location selected. The problem is to select a location for the facility that is fair to all concerned parties. Classical utilitarianism seeks to maximize the total social welfare, i.e., the sum of the utilities. Egalitarianism on the other hand seeks to equalize the utilities to all agents, and whenever this is not feasible exactly, to approximate it by maximizing the minimum utility (and within the solutions that achieve this, select the one maximizing the second smallest utility, etc.). Another solution, proposed by Nash, seeks to maximize the *product* of the utilities. Here of course one needs to assume that the utilities are non-negative. Moulin goes over several examples exploring the difference between these various solutions, and illustrating contexts in which each most appeals to one's innate sense of fairness.

The conflicting fairness principles of compensation, reward and fitness are discussed at length in chapter 2, together with the notion of Pareto efficiency. This chapter also looks at simple instances of the utility function model, such as when all the agents have a common utility function. Chapter 3 treats the utility function model in more detail, looking at broader classes of utility functions, as well varying the social objectives of the benevolent dictator.

One criticism of the utility function model is that it assumes that the happinesses of different agents are comparable. While this may be reasonable when the measurement of happiness is via something tangible like the actual distance to a facility or the amount of money involved in a monetary transaction, it is harder to justify when we are talking about an individual's appreciation of art or preference for certain political or religious beliefs. Moreover, a utility function imputes intensities to an agent's preferences, which may not be justified. A scheme based on utilities rather than just the order they impose on the outcomes must capture these intensities and, in its pursuit of fairness, will have the effect of rewarding the fanatics at the expense of those with a natural tendency towards happiness. To avoid this, economists study models in which each agent only specifies her preference relation (a total ordering) on the set of all possible outcomes. Such a model may be thought of as a scheme in which the agents *vote* on the "best" of all the candidate outcomes.

Voting schemes

Consider an election in which the agents must vote for and select one out of a number of candidates. When there are only two candidates the "best" is easily defined to be the candidate preferred by more agents (breaking ties at random). When there are more candidates, however, things are not that simple. A simple vote count based only on each voter's top choice could lead to more than half the population being unhappy with the winner. Thus we need a scheme that takes into account the entire profile of preferences for each voter. Two such schemes were proposed about two centuries ago by the French philosophers Borda and Condorcet.

Borda's scheme gives each candidate a score based on their rank in each voter's profile. If there are n candidates, each candidate receives a score of $n - i$ for every voter that ranks it in position i. The cumulative scores give rise to a transitive relation on the candidates. The winning candidate is the one with the highest score (ties broken at random).

In Condorcet's scheme we construct the tournament graph on all candidates. This is a directed graph on all the candidates with an edge from i to j if the majority prefers candidate i to candidate j (in the induced game where those are the only two candidates). The winner of the tournament is the candidate that beats all other candidates (i.e., has in-degree zero). Unfortunately there may not be such a candidate, since the relation induced by the tournament need not be transitive (i.e., the graph may contain directed cycles). In this case there is no clear Condorcet winner and, although there are a number of proposed methods of breaking cycles to select a winner, none is completely satisfactory. By contrast, Borda's scheme always elects a winner.

One advantage of Condorcet's scheme over that of Borda is that in the former, the relation between any two candidates is not affected by their standings relative to a third. The aggregate relation is said to be independent of irrelevant alternatives. In the Borda scheme however, the disappearance of a minority candidate (one who has no chance of winning) can change the outcome of the election, as in the following example.

There are three candidates A, B and C and seven voters. They are ranked A, C, B by two voters, A, B, C by one voter and B, A, C by four voters. Then A wins with a score of 10, followed by B with a score of 9 and C with a score of 2. But if C is struck off the ballot, then B wins with a score of 4 compared to A's 3.

Because it is independent of irrelevant alternatives, Condorcet's scheme is also robust against strategic misreporting of preferences, whereas in the scoring scheme, an agent whose true top choice is a candidate who stands no chance has an incentive to misreport their second choice as the top one, as this could possibly tip the balance in favour of their second choice.

Thus there are reasons to prefer either scheme over the other. In fact, this is unavoidable: a famous theorem of Arrow states that no preference aggregation scheme for at least three candidates can both be independent of irrelevant alternatives and always elect a winner, with the sole exception of dictatorship (where the aggregate is just the preference of a single special agent).

Chapter 4 of the text is devoted to voting schemes. The author introduces both kinds of voting schemes and gives many examples to illustrate their relative merits and drawbacks, in various situations. He also presents a class of preference relations which guarantee the existence of a Condorcet winner. Finally, he discusses Arrow's theorem and its implications.

Sharing the Commons

In many situations a common resource ("commons") is used jointly by a number of agents and we encounter the problem of regulating the use of the resource to be fair and efficient. Examples of this include the division of surplus (or loss) in a joint venture (such as a law firm or cooperative), cost sharing in shared residences or condominium buildings and the regulation of traffic on road networks.

Consider the problem of providing power to a number of townships. We have to build a network of power lines that connects all of the towns to the generating plant. Suppose we have already determined the cheapest way to build such a network and what remains is the problem of splitting the costs among the various towns. A first idea might be to to simply split them evenly, but this seems unfair if, say, one of the towns is much further than the others. For concreteness, say towns A, B and C and the power plant lie on a straight line, in that order. Then a fair solution is for

A to pay for the cost of the wiring from A to B, A and B to evenly share the cost from B to C and all three evenly share the cost of connecting C to the power plant. Thus the cost of each segment is shared equally by all the towns using it. This solution is based on the Shapley value which charges each individual his expected marginal cost (i.e. the cost to be included in the game) over the preceding agents in a uniformly random ordering.

When a resource is not large enough to support the full demands of all the agents, outcomes produced by the strategic actions of individual agents (i.e., Nash equilibria) may be *Pareto inefficient.* Everyone is familiar with the phenomenon of rush hour traffic on the freeway increasing to point where it moves slower than the "slow roads" (if at all). The phenomenon of inefficient sharing of resources due to strategic play is called the "tragedy of the commons," and arises in diverse other contexts.

Chapter 5 is devoted to the concept of the Shapley value. It is motivated through several illustrative examples and formally defined. The author also presents an axiomatization, showing that the Shapley value is the only function that satisfies all of certain desirable fairness criteria. In chapter 6, the author outlines and compares various schemes for regulating the use of the commons, including one based on the Shapley value and discusses scenarios in which each is the preferred one.

Fair division and envy

Suppose we have a cake that is to be divided (fairly, of course) between three children. If the cake is homogeneous then we simply cut it into three identical pieces (let's assume that this is possible) and give each child one of them. However suppose that the cake is not homogeneous, and that each child has different tastes. One likes the pink sugar roses, the second prefers the swirls of whipped cream, while the third wants more cake and less frosting. If any child perceives another child as having received a better piece, the children will quarrel. The problem now becomes one of not merely equalizing the pieces (which is not very meaningful for a non-homogeneous cake anyway) but of dividing the cake so that each child will prefer her own piece to those of the others. In other words no one will be envious of anyone else. The situation becomes even more complicated when instead of a cake, a collection of indivisible objects (such as toys) is to be divided.

The pros and cons of a few approaches to this very difficult problem are the subject of chapter 7.

General impression

This is a well thought-out book. It is aimed at advanced undergraduates in economics but should also be quite accessible to most computer science students. The text is fairly self-contained, although a certain degree of mathematical sophistication and familiarity with microeconomic thinking will be helpful to the reader.

One thing I really liked is that each concept is developed through examples, which although simple, are nevertheless somewhat plausible instances of real-world problems.

The focus of the book is primarily on the concept of fairness and theoretical solutions to problems of fair division such as those mentioned in the preceding sections. Although algorithms and computational issues are mentioned, they are not a major concern in this text, and are mainly relegated to the exercises.

This is intended as an introductory text, and for the most part, axiomatizations and formal proofs of results have been omitted (although the book does include extensive pointers to the literature). A large fraction of the book is devoted to exercises, ranging from straightforward applications of the concepts discussed to more interesting and involved problems. A couple of caveats about the readability of the book: many of the figures are insufficiently labelled or captioned, making them a challenge to decipher, and simple points seem to be belaboured in some places, while in other places a desire to keep the discussion informal has led to sloppiness. Nevertheless, this book should be valuable to anyone interested in learning about algorithmic game theory and electronic commerce.

Review[4] of

Algorithms: Sequential, Parallel, and Distributed
Authors: Kenneth A. Berman and Jerome L. Paul
Publisher: Thomson Course Technology, 2005
$84.95, Hardcover, 962 pages
Reviewer: William Schmeister

Introduction

Algorithms: Sequential, Parallel, and Distributed is an introduction to algorithms meant to provide a solid foundation for classical theory of algorithms. The target audience of the book, as stated in the preface, along with the content of the book, is the upper division undergraduate or the first-year graduate student.

A wide range of algorithms are covered in the book, not only standard sequential algorithms but also a wide array of modern parallel algorithms. The algorithms in the book range from optimization of exponential algorithms to an introduction of complex NP-Complete problems. Included in this coverage is one very commonly used algorithm, one we all may use without realizing, the PageRank algorithm that was developed by Larry Page and Sergey Brin, the cofounders of Google.

Summary

The authors have organized the text into five parts: Introduction to Algorithms, Major Design Strategies, Graph and Network Algorithms, Parallel and Distributed Algorithms, and Special Topics.

Part I Introduction to Algorithms

The book begins with an introduction of how algorithms have changed from ancient to modern times. Some examples include how inefficient mathematical algorithms from ancient civilizations have evolved and how their modern counterparts have become quite efficient with the help of computers. Pseudo code is provided for both the ancient and modern methods and even some stages in between. The first chapter finished with some formal definitions and introductions to the use of algorithms in modern times.

Fundamentals of design and analysis is covered in the next chapter. The ever important aspect of recursion is introduced in this section, along with the importance of how data structures design affects the choice of algorithm design. Performance is introduced as a term to compute the complexity and running times of an algorithm; various search algorithms are used to illustrate complexity. Other modern techniques that are introduced are serial, parallel, internet, distributed and quantum computing.

The authors of the book assert, as most algorithm designers would agree, that more important than the efficiency of an algorithm is the correctness of that algorithm. The authors introduce mathematical induction to prove the correctness of an algorithm and provide many example proofs

[4]©2006 William Schmeister

for previously examined algorithms. At this point, Hard and NP-Complete problems are also introduced.

To finish the first section of the book, three chapters are presented to allow a more in depth coverage of three important fields in the study of algorithms: Trees, sorting, and average complexity. The trees chapter presents a number of different tree structures, how they may be applied, and various algorithms used within the trees such as traversal, insertion, removal, and balancing. Sorting provides a few more algorithms, including the shell sort, bingo sort, and radix sort; it also includes techniques to sort lists that cannot reside in internal memory for one reason or another. Average complexity covers examples to determine complexity of the previously described algorithms.

Part II Major Design Strategies

Part II is only about a hundred pages long; yet, it holds some very important aspects of designing algorithms. The first technique introduced and covered is the greedy method. The greedy method is exemplified by the change making problem and the knapsack problem. The final and longest section explains Huffman codes. Huffman codes are a type of greedy method that compresses data.

The second major design strategy is the divide and conquer paradigm. This paradigm is expressed in a number of examples: symbolic algebraic operations on polynomials, multiplication of large numbers, matrix multiplication, and some examples in computational geometry. The computational geometry examples specifically cover the closest pair and the convex hull problems.

The third chapter in this part of the book describes a number of problems solvable with dynamic programming. Optimal parenthesization, optimal binary search trees, and longest common subsequence are used as examples.

Backtracking and branch-and-bound is the final and longest chapter in this part. One of the aspects in this section worth noting is that solutions are presented in both recursive and non-recursive forms. On the other hand, upon testing some of the examples, the pseudo code did not function correctly. The correct solutions were not in the books errata page at the time of reading, but many corrected algorithms were found on professors web pages who have used the book.

Part III Graph and Network Algorithms

The first chapter of this part is dedicated to graphs and digraphs. The majority of the chapter defines graphs and di-graphs (directional-graphs); it is followed by a number of operations that can be performed on a graph. Searching and sorting algorithms are the bulk of the operations. The list of algorithms include: Depth-first search, breadth-first search, and topological sorting.

Once the user has an understanding of graphs, the text continues with minimum spanning trees and shortest path algorithms. Minimum spanning tree algorithms are expressed through two strategies: Kruskal's algorithm and Prim's algorithm. Shortest path algorithms are then described and viewed through three famous algorithms: Bellman-Ford, Dijkstra, and Floyd's algorithms.

Graph connectivity or fault tolerance of networks is the next topic covered by Berman and Paul. Strongly connected components are defined through a number of propositions; then, an algorithm and di-graph are provided to find these components. Articulating points and bi-connected components follow strongly connected components. These two new systems are presented to the reader by the same techniques as before; informal definitions, theorems, and propositions followed by an algorithm and pseudo code.

The final chapter covers matching and network flow algorithms. Perfect matching and the marriage problem are discussed with a solution using the Hungarian algorithm. Maximum perfect matching is outlined in a weighted graph using the Kugn-Munkres algorithm. Capacitated networks and maximum flow are also discussed; the book covers many example networks and techniques to determine the maximum flow in a network.

Part IV Parallel and Distributed Algorithms

Part IV introduces the reader to the concept of parallel and distributed algorithms. This section should be a requirement for young and new computer scientists and programmers because of the frequency of distributed processes. Parallel strategies, internet, distributed computation, and distributed network algorithms are all covered in this impressive section.

The first chapter in this part provides an introduction to parallel algorithms and architectures. The chapter contains a wealth of information and important concepts for the use of distributed algorithms. It opens with many considerations that an algorithm designer must consider when designing an algorithm; some of these considerations include instruction type (single instruction vs. multiple instructions on parallel computers), number and types of processors, and shared memory. From that foundation, the book continues with examples on how to use these attributes to exploit for the systems benefit.

The introductory chapter continues with the costs and benefits to parallel systems. Formulas are provided to weigh the speedup and the cost of using a parallel algorithm. Amdahl's law expressing the upper bound for the speedup achievable by any parallel algorithm rounds out this section of the chapter. Parallel algorithms for sorting is the last topic covered in this introductory chapter.

With the introduction to parallel algorithms complete, the book continues with strategies for the design of parallel algorithms. Parallel prefix computation is covered first in this section; the text states that though prefix sums are inherently sequential, in fact, [they are] highly parallelizable. The text continues with examples of how they can be parallelized. Building on the parallel prefix computation, the knapsack problem is solved as a parallel algorithm. Pointer jumping and matrix operations are the last two topics that are covered under the design strategies chapter.

Internet algorithms are covered in the following chapter. Search engines are introduced by covering the original PageRank algorithm from the founders of Google and the HITS (Hyperlink Induced Topic Search) developed by Jon Kleinberg of IBM. Formulas that determine how each web page is ranked are included, unfortunately, only the HITS algorithm contains pseudo code, though the implementation of the PageRank algorithm should be trivial.

The text claims that internet algorithms require good hashing functions to efficiently retrieve data from a database. The authors provide two good hashing functions, chaining and open addressing that prevent collisions of hash codes. The authors provide internet users with methods for retrieving the content efficiently and quickly. The last section of this chapter is dedicated to the idea that information should be secure and non-repudiatable; this refers to using cryptography and digital signatures.

The next chapter covers the concept of algorithms distributed through networked systems. The introduction to this chapter includes the differentiation and definitions of distributed network terms, such as distributed algorithm, computational grid, and Beowulf clusters. The text focuses on weakly synchronized processes that generally communicate using message passing. That being said, the first major topic in the chapter is message passing and types of message passing, including synchronous, asynchronous and others. A number of other types of computation algorithms are included, along with some concerns for those algorithms.

The master-worker paradigm is covered with example code for matrix multiplication. After the master-worker parallel algorithm is described, the shared-task paradigm is covered. The shared-task paradigm is where one worker completes its task and that available worker helps another worker complete its task. The final section of this chapter uses the techniques previously learned to implement a distributed depth-first search.

The concluding chapter is dedicated to network algorithms. The first examined algorithm is the ring network that is the basic model of the token ring. There is an in-depth discussion of leader election for ring networks. The second network type is the broadcasting and breadth-first search paradigm. The next algorithms are the shortest path, all-pairs shortest path, minimum spanning trees, and the asynchronous model.

Part V Special Topics

The last part of the book is the longest; yet, it could be much longer. The breadth of the material is quite large and potentially complex; an in-depth discussion would not be appropriate in an introductory book. Therefore, the chapter length is quite proper. The topics in this part include: String matching, balanced search trees, Fourier transforms, game trees, probabilistic randomized algorithms, lower-bound theory, np-complete, and approximation algorithms. All of these topics are covered in a mere two-hundred and thirty pages.

The first topic, String Matching and Document Processing, is a commonly used situation. The text discusses and presents an example of a nave algorithm for string matching. The nave method is quickly refuted with formal algorithms for solving string matching: Knuth-Morris-Pratt, Boyer-Moore, and Karp-Rabin. Approximate string matching is introduced before proceeding into tries and suffix trees.

Balanced search trees are covered in an in-depth format. The dictionary problem is used as an example of why balanced trees should be utilized. Rotation, as the principle for keeping a tree balanced, is demonstrated. Also, algorithms are provided with examples for Red-Black trees and B-Trees

One of the most famous divide-and-conquer algorithms, Fourier transforms, has a chapter in this part. Discrete Fourier transforms are introduced; this rapidly progresses into the fast Fourier transforms. A recursive solution to the Fourier Transforms is covered; it is followed by an iterative approach.

Next, Heuristic search strategies are covered. Heuristic search strategies, such as A* (A-star), are common search algorithms; they are used in artificial intelligence and games. The use of heuristic search strategies is illustrated by the 8-puzzle game, a smaller version of the 15-puzzle, invented by Sam Lloyd. Once the problem set is explained in plain English, A* is put to use by building a heuristic. A* is then used to solve the 8-puzzle and to find the shortest path. A* takes on a modification for branching and bounding with the use of a bounding function to meet the objective. Game trees are examined next, describing how heuristic search strategies are used to provide a system with the means to make perfect plays in a game.

Pointing out that all of the algorithms to date have been deterministic, the text proceeds to describe some of the available non-deterministic algorithms. Non-deterministic algorithms are also known as probabilistic and randomized algorithms. A few pages of text are written to describe how to randomize some deterministic algorithms. Monte Carlo and Las Vegas algorithms are presented; though the book does not state this, these techniques are very important to engineers trying to solve problems as they are heavily used in Kalman filters and similar algorithms. The probabilistic chapter comes to an end after brief discussions on probabilistic numerical algorithms and probabilistic parallel algorithms.

Lower-bound theory is discussed. The object of this section is to discuss various techniques for determining good lower-bound estimates for the worst-case and average complexities of problems. Basic terminology and techniques are presented to the reader pertaining to a number of different areas. Decision and comparison trees, adversary arguments, and parallel algorithms are all discussed with respect to lower-bound theory.

NP-Complete problems are re-visited, beginning with the comparison between P and NP problems. A short list of NP problems are mentioned and described; these problems include clique, sum of subsets, and Hamiltonian cycle. Reducibility, along with Cooks theorem, is discussed here. A few sample NP-Complete problems are discussed, and co-NP, NC, and P-complete classes are discussed.

If there is no known solution to a problem within polynomial time, an approximation to the answer may be in order. That is the topic of the last chapter in the text. A number of typical problems are presented that are appropriate to this solution; they include the traveling salesman, bin packing, Steiner tree, and facility location problems

Appendices

After all of the regular chapters are complete, the appendices span more than seventy pages. The appendix pages are well worth reading and should be reviewed. Mathematical notation provides some needed background, if the reader feels deficient in basic mathematical notations. Linear data structures are reviewed with definitions of stacks, queues, linked lists, and similar structures. Interpolating asymptotic behavior is briefly discussed, which is followed by random walks in digraphs and elementary probability theory. Examples of message-passing interface code are added to assist the reader in understanding distributed system message passing. The final appendix helps describe the pseudo code nomenclature.

Opinion

Overall, I think this book has some really good components. Software engineers need to have a wide array of weapons in their arsenal, and everything in this book should be part of that arsenal. A good software engineer needs to be at least familiar with all techniques described in this book. I think that grouping the book into these five parts was advantageous; readers needing a specific refresher may only need to read the appropriate part. Also, the inclusion of parallel and distributed algorithms is necessary in a world where the system of systems architecture is becoming more pronounced, and it is a technique all software engineers should be aware of.

My biggest complaint about this book is the pseudo code. I found many instances where the code was just plain wrong; in some instances, the implementation was impossible, and if it was implemented, the results were incorrect. Also, many of the problems that I found were not corrected in the errata page. Though I corrected some of the pseudo code myself, I finally quit coding all together because of the probability that the code would not work once implemented. I did find that other professors have obviously had the same problem because some had actually written their own errata pages for the book with correct code.

Review[5] of

Algebraic Complexity Theory

Authors: Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi

Springer 1997, 618 pages

Reviewed by Anthony Widjaja `twidjaja@cs.mu.oz.au`, University of Melbourne

1 Introduction

Algebraic complexity theory can be defined to be the study of the minimum number of operations required to solve various computational problems that are algebraic in nature. Examples of such problems include polynomial evaluation, polynomial multiplication, and matrix multiplication. Unlike the field of computer algebra, whose goal is to establish tighter *complexity upper bounds* for

[5]©Anthony Widjaja, 2006

algebraic problems by designing faster algorithms, algebraic complexity theory aims to determine the complexity of algebraic problems with an emphasis of proving *complexity lower bounds*.

Algebraic complexity theory grew rapidly since 1954 when Ostrowski inquired about the optimality of Horner's rule for polynomial evaluation. Although there are excellent surveys (e.g. [6, 10, 11]) and research monographs (e.g. [2]), no one had previously attempted to produce a systematic treatment of the theory. The book Algebraic Complexity Theory (ACT) is the first such attempt. As the authors of the book claim, ACT is intended to serve as a comprehensive textbook well-suited for beginners at the graduate level, as well as a reference book for researchers already working in the field, and requires only some basic (abstract) algebra. I review this book mainly from the point of view of beginners at the graduate level who have been exposed to basic algebra and other basic pure mathematics (such as topology, mathematical analysis, graph theory, and combinatorics), but not to algebraic complexity theory.

2 A concrete example

Many of the readers are probably unfamiliar with algebraic complexity theory. Thus, it is perhaps best to illustrate the theory by means of a simple concrete example. Let R be an arbitrary ring (usually infinite) and n a positive integer. Consider the problem EXP of computing X^n given $X \in R$ as an input. We are interested in the minimum number of steps required to compute the problem. In order to determine this number, it is important to first specify the model of computation.

For this example, we shall adopt the following computation model. Our machine is capable of multiplying two "intermediate results" in one step (the initial intermediate result being X). The result of each multiplication is also considered an "intermediate result". We say that the machine *computes* the problem EXP if X^n is among the intermediate results given the input X. Let $l(n)$ be the minimum number of multiplications required to compute X^n from X. We want to determine the rate of growth of l.

Trivially, we have $l(n) \leq n-1$ because in each step one can multiply the last intermediate result by X yielding the sequence $(X, X^2, \ldots, X^n)$ of intermediate results. We can improve this upper bound by making use of the "fast exponentiation" algorithm. This recursive algorithm works as follows. If $n = 2m$, we first compute X^m and then square this to obtain X^n. If n is odd, we first compute X^{n-1} and obtain X^n by multiplying X^{n-1} by X. The base case occurs when $n = 1$, in which case we do nothing (as X is always part of the intermediate results). This algorithm shows that

$$l(n) \leq \lfloor \log n \rfloor + w_2(n) - 1 \leq 2 \log n$$

for $n \geq 2$ where w_2 denotes the number of nonzero coordinates in the binary representation of n, and log is the base-2 logarithm function. To see this result, simply scrutinize the above algorithm and observe that, for $m > 0$, we have $l(2m) \leq l(m) + 1$, $l(2m+1) \leq l(2m) + 1$, and $l(1) = 0$. With $w_2(2m) = w_2(m)$ and $w_2(2m+1) = w_2(2m)+1$, the claim follows immediately from a simple induction.

Can we improve this upper bound? The answer is 'yes', but not by a significant factor. In fact, it is the case that $l(n) \geq \lceil \log n \rceil$. This lower bound can be achieved by a simple degree argument: the maximum of the degrees (of X) obtained in a step can at most double the maximum of the degrees in the current intermediate results. This shows that $l(n) \sim \log n$.

Although this example represents the task of algebraic complexity theory in general, I by no means want to imply that the theory is simple. In fact, many complexity lower bounds derived in ACT require the use of sophisticated mathematics such as algebraic topology, algebraic geometry, and combinatorics, to name a few.

3 Summary of Contents

Note: our summary is not intended to be exhaustive. For the complete table of contents, see [1].

The book consists of 21 chapters. The first chapter is an informal introduction to algebraic complexity theory. It motivates the study of algebraic complexity theory, and provides a cursory glance on the rest of the book. The rest of the book is divided into five parts.

Part I, consisting of chapters 2 and 3, concerns fundamental algorithms for several problems in algebraic complexity theory. This part of the book requires only some algorithmic intuition. The authors intentionally postpone the formal definition of the models of computation until chapter 4 in favor of easier access to the results. Chapter 2 deals with the design and analysis of algorithms for problems concerning symbolic manipulation of polynomials and power series. These problems include the multiplication, inversion, division, and composition of polynomials and power series. This chapter presents several clever algorithms such as the Fast Fourier Transform (FFT) algorithm for polynomial multiplication over "nice" fields, the Schönhage-Strassen algorithm for polynomial multiplication over arbitrary fields, and the Brent-Kung algorithm for the composition of power series. Chapter 3 focuses on algorithms within the branching model. This chapter starts off with the fast Knuth-Schönage algorithm for computing the greatest common divisor of univariate polynomials, and ends with the application of VC-dimension and the theory of epsilon nets to proving that certain NP-complete problems are solvable by "nonuniform" polynomial-time algorithms in this model. Note that this is not the same as non-uniform P as defined by complexity theorists normally.

Part II, consisting of chapters 4 – 7, discusses some elementary lower bound results in algebraic complexity theory. Chapter 4 formally defines the models of computation that are used in algebraic complexity theory such as *straight-line programs*, and *computation trees*. In addition this chapter also presents the first complexity lower bound in this book, the *Dimension Bound*, which is then used to derive some complexity lower bounds for problems defined in Part I. Chapters 5 and 6 contain techniques for showing the optimality of Horner's rule for polynomial evaluation in the generic case. Horner's rule asserts that a polynomial p of degree n can be evaluated using n multiplications and n additions. Ostrowski's conjecture reads: (1) evaluating polynomials of degree n requires at least n multiplications regardless of the number of additions or subtractions used, and (2) evaluating polynomials of degree n requires at least n additions or subtractions regardless of the number of multiplications and divisions used. Chapter 5 contains the transcendence degree argument of Motzkin and Belaga for proving (2). On the other hand, when "preconditioning of the coefficients" of p is allowed, Horner's rule is not optimal. For example, when $p \in K[X]$ for a fixed algebraically closed field K, we need only $n + \lfloor n/2 \rfloor + 3$ arithmetic operations for computing p. Chapter 6 focuses on Pan's substitution method, which is used for proving (1). The method is then generalized and, in particular, applied to the problem of computing continued fractions. Chapter 7 covers two differential methods. The first is Strassen's method for "avoiding divisions". Loosely speaking, this method transforms a straight-line program for computing a set of rational functions to a division-free straight-line program for the coefficients of the Taylor series of these functions. In particular, this method is applied to proving that divisions do not help for the computation of a set of quadratic polynomials. This result initiated the theory of *bilinear complexity*, which will be explored further in chapter 14. The second method, due to Baur and Strassen, can be used to show that computing a multivariate function f and all its first-order partial derivatives takes at most four times the amount of time of computing f itself. This result, referred to in this book as *Derivative Inequality*, has some very important consequences in proving complexity lower bounds. For example, in this chapter we will see one implication: inverting a matrix is about as hard as computing the determinant.

Part III, consisting of chapters 8 – 12, deals with complexity lower bounds for systems of polynomials of high degree. Some of these lower bounds are proved using techniques from algebraic geometry and algebraic topology. Chapter 8 concerns Strassen's degree bound, which is a fundamental tool for proving nonlinear complexity lower bounds in algebraic complexity theory. Strassen's degree bound relates the complexity of computing a finite set of rational functions to the geometric degree (in the sense of algebraic geometry) of the graph of the associated rational map. We will see here that the degree bound can be applied to proving that some algorithms described in chapter 2 (such as the algorithm for computing the n-variate elementary symmetric polynomials) are essentially optimal. This chapter also contains a reasonably self-contained introduction to the relevant notions from algebraic geometry. Chapter 9 develops methods for proving complexity lower bounds for the computation of specific polynomials (compare with chapter 5 and 6). Chapter 10 presents a degree bound for the computation trees. In particular, the result is applied to proving that the Knuth-Schönage algorithm (discussed in chapter 3) is optimal. Chapter 11 presents Ben-Or's complexity lower bound for semi-algebraic membership problems. The chapter also contains some applications to the real knapsack problem and several problems in computational geometry. Chapter 12 proves Grigoriev and Risler's lower bound on the (additive) complexity of a univariate real polynomial in terms of the number of its real roots.

Part IV, consisting of chapters 13 – 20, concerns the problem of computing a finite set of multivariate polynomials of degree at most two. Chapter 13 deals with the problem of computing a finite set of linear multivariate polynomials. More precisely, given a matrix $A \in K^{m\times n}$ and a vector $X = (X_1, \ldots, X_n)^{\mathrm{T}}$ of indeterminates, we want to compute AX. This problem subsumes many smaller problems of practical importance such as that of computing the discrete Fourier transform, which is important for polynomial multiplication. An exact complexity for this problem is derived for the generic case, while some tight lower bounds for this problem are derived for some special matrices A (such as the discrete Fourier transform and Vandermonde matrices). Chapters 14 – 20 contains a systematic treatment of the theory of bilinear complexity, which concerns the complexity of bilinear mappings. In some sense, one may think of this theory as a general framework within which to talk about the problem of matrix multiplication. Chapter 14 is an introduction to the theory of bilinear complexity. Chapter 15 is devoted to the problem of determining the asymptotic complexity of matrix multiplication. It presents an intricate proof that matrix multiplication can be solved in $O(n^{\omega})$ where $\omega < 2.39$, which is a significant improvement for the upper bound of ω (the first upper bound for ω, due to Strassen, was 2.81). This bound is not the best known. Coppersmith and Winograd [4] have $O(n^{2.376})$ algorithm. The authors wisely chose not to present this algorithm since it is rather complicated.

It is shown in chapter 16 that some problems in computational linear algebra — such as matrix inversion, computation of the determinant, and LUP-decomposition — are as hard as matrix multiplication. Chapter 16 also contains a nice application of matrix multiplication algorithms to the problem of computing the transitive closure of a graph (in the sense of graph theory). Chapter 17 concerns methods for proving complexity lower bounds for the computation of matrix multiplication and, in general, bilinear maps. In chapter 18, there is an interesting connection of the theory of bilinear complexity (over finite fields) and coding theory. Chapters 19 and 20 study tensorial rank with respect to special classes of tensors and generic tensors, respectively.

The first twenty chapters of this book discuss problems with "tame" complexities (polynomials of small degree). The last part of this book, which consists of chapter 21, studies several problems for which the known complexity upper and lower bounds differ substantially (e.g. the lower bounds are polynomials of small degree, while the upper bounds are exponential). Examples of such problems include the problems PER and HC of, respectively, computing permanents and Hamiltonian cycle polynomials. This chapter starts by defining Valiant's algebraic complexity classes **VP** and **VNP** — the nonuniform algebraic analogs of the complexity classes **P** and **NP**. We will see here that the problems PER (over any field of characteristic different from two) and HC are **VNP**-complete. This chapter also discusses Valiant's nonuniform algebraic analog of the **P** vs. **NP** problem, which asks whether $\mathbf{VP} \neq \mathbf{VNP}$ (over any field). The topics covered in this chapter is further explored in another book [3] by one of ACT's authors.

4 Opinions

This book is certainly the most complete reference on algebraic complexity theory that is available hitherto. Many current research topics are covered except for subjects such as parallel and randomized algorithms, and time-space tradeoff, which are intentionally excluded. This book is also well-organized. Each chapter is made as independent of the other chapters as possible allowing quick-and-easy access to the results of interests to the readers. The dependencies among the chapters are delineated in a dependency graph in the book preface. Some open problems are also mentioned at the end of each chapter, which helps the readers to get up to speed with the current state of the art. Also, superb bibliographical and historical notes are given at the end of each chapter.

Although a few proofs in this book are somewhat too concise for my liking, the book is overall very readable. In many cases the authors took an extra care when explaining difficult concepts. However, I have to admit that the book requires much more than the knowledge of basic algebra. For example, some parts of the book require the readers to be familiar with topology. Moreover, some of the algebraic concepts, with which the readers are expected to be familiar, are not part of basic algebra. [At the very least, these concepts were not taught in my first course on abstract algebra, which use [7, 8] as textbooks]. Fortunately, there are some excellent textbooks on abstract algebra (e.g. [5, 9]) that should suffice the algebra prerequisites of ACT. The authors could probably make the book more user-friendly for beginners by briefly recalling some of these definitions (and their associated basic facts) in the appendix. On the other hand, this book would most certainly make a great textbook for a graduate course on algebraic complexity theory. [In fact, I think this is the best way for beginners to learn from the book.] ACT has over 350 exercises (some being open problems). Although I found most of the exercises very challenging, I am glad that the authors mention the sources of the exercises at the end of each chapter from which I can find solutions. To the best of my knowledge, no online errata for this book is maintained. Nevertheless, the book contains surprisingly few errors, which are mostly typographical and can be easily spotted.

In conclusion, any researchers already working in the area should own a copy of this book. Although the readers who have not been working in the area may find the book somewhat too challenging for a self-study, this book would nonetheless be suitable for a graduate course on algebraic complexity theory. Provided the supervision or instruction of a teacher (and the willingness to read up on the background materials from other sources), beginners at the graduate level who have been exposed to undergraduate pure mathematics would find this book accessible.

References

[1] ACT's website. `http://math-www.uni-paderborn.de/pbuerg/agpb/act/act.html`. 2005.

[2] A. Borondin and I. Munro. The Computational Complexity of Algebraic and Numeric Problems. American Elsevier, 1975.

[3] P. Bürgisser. Completeness and Reduction in Algebraic Complexity Theory. Springer-Verlag, 2000.

[4] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions, *Journal of Symbolic Computation*, 1990. Earlier version in 1987 STOC.

[5] D. S. Dummit and R. M. Foote. Abstract Algebra. Wiley, 2003.

[6] J. von zur Gathen. Algebraic Complexity Theory. *Annual Review of Computer Science*, volume 3, pages 317–348, 1988.

[7] B. Hartley and T. Hawkes. Rings, Modules, and Linear Algebra. Chapman & Hall, 1970.

[8] T. W. Hungerford. Abstract Algebra: An Introduction. Brooks Cole, second edition, 1996.

[9] T. W. Hungerford. Algebra (Graduate Texts in Mathematics). Springer-Verlag, 1997.

[10] N. Pippenger. Algebraic Complexity Theory. *IBM J. RES. DEVELOP.*, volume 25, no. 5, September 1981.

[11] V. Strassen. Algebraic Complexity Theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume A*, chapter 11, pages 634 – 672, Elsevier Science Publishers B. V., Amsterdam 1990.