Approximation Algorithms *for* NP—Hard Problems

NP 难解问题的近似算法

edited by Dorit S. Hochbaum

ITP
世界图书出版公司

APPROXIMATION ALGORITHMS FOR NP-HARD PROBLEMS

Edited by DORIT S. HOCHBAUM
University of California — Berkeley

PWS PUBLISHING COMPANY

I(T)P
An International Thomson Publishing Company

世界图书出版公司

北京 · 上海 · 广州 · 西安

书　　名：Approximation Algorithms for NP-hard Problems
编　　者：D. Hochbaum
中 译 名：NP 难解问题的近似算法
出 版 者：世界图书出版公司北京公司
印 刷 者：北京中西印刷厂
发　　行：世界图书出版公司北京公司(北京朝内大街 137 号　100010)
开　　本：大 32　　印张：19.5
版　　次：1998 年 3 月第 1 版　　1998 年 3 月第 1 次印刷
书　　号：7-5062-3630-3
版权登记：图字 01-97-1451
定　　价：93.00 元

LIST OF CONTRIBUTORS

Sanjeev Arora
Computer Science Department
Princeton University
35 Olden Street
Princeton, NJ 08544
arora@cs.princeton.edu

Marshall Bern
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304
bern@parc.xerox.com

Ed G. Coffman
Room 2D-150
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974-2070
egc@research.att.com

David Eppstein
Department of Information
and Computer Science
University of California at Irvine
Irvine, CA 92717-3425
eppstein@ics.uci.edu

Mike R. Garey
Room 2D-150
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974-2070
mrg@research.att.com

Michel X. Goemans
Department of Mathematics
Room 2-382
MIT
Cambridge, MA 02139
goemans@math.mit.edu

Leslie Hall
Mathematical Sciences
Maryland Hall
Johns Hopkins University
3400 North Charles Street
Baltimore, MD 21218-2694
leslie@noether.mts.jhu.edu

Dorit S. Hochbaum
Department of Industrial Engineering
& Operations Research
Etcheverry Hall
University of California
Berkeley, CA 94720-1777
dorit@hochbaum.berkeley.edu

Sandy Irani
Department of Information
and Computer Science
University of California at Irvine
Irvine, CA 92717-3425
irani@ics.uci.edu

Mark Jerrum
Department of Computer Science
University of Edinburgh
King's Buildings
Mayfield Road
Edinburgh EH9 3JZ
U.K.
mrj@dcs.ed.ac.uk

David S. Johnson
Room 2D-150
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974-2070
dsj@research.att.com

Anna R. Karlin
Department of Computer Science
and Engineering
FR-35
University of Washington
Seattle, WA 98195
karlin@cs.washington.edu

Samir Khuller
Computer Science Department
University of Maryland
College Park, MD 20742
samir@cs.umd.edu

Carsten Lund
AT&T Bell Laboratories
Room 2C-324
600 Mountain Avenue
Murray Hill, NJ 07974
lund@research.att.com

Rajeev Motwani
Department of Computer Science
Stanford University
Stanford, CA 94305-2140
rajeev@cs.stanford.edu

Joseph (Seffi) Naor
Department of Computer Science
Technion—Israel Institute of Technology
Haifa 32000
Israel
naor@cs.Technion.ac.il

Balaji Raghavachari
Computer Science Department
University of Texas at Dallas
Richardson, TX 75083-0688
rbk@utdallas.edu

Prabhakar Raghavan
IBM, Box 218
Yorktown Heights, NY 10598
pragh@watson.ibm.com

David B. Shmoys
204 E&TC Building
School of Operations Research
and Industrial Engineering
Cornell University
Ithaca, NY 14853-3801
shmoys@cs.cornell.edu

Alistair Sinclair
Computer Science Department
University of California
Berkeley, CA 94720
sinclair@cs.berkeley.edu

David P. Williamson
IBM TJ Watson Research Center
Room 33-219
P.O. Box 218
Yorktown Heights, NY 10598
dpw@watson.ibm.com

INTRODUCTION

Approximation algorithms have developed in response to the impossibility of solving a great variety of important optimization problems. Too frequently, when attempting to get a solution for a problem, one is confronted with the fact that the problem is *NP*-hard. This, in the words of Garey and Johnson, means "I can't find an efficient algorithm, but neither can all these famous people" ([GJ79] p. 3). While this is a significant theoretical step, it hardly qualifies as a cheering piece of news.

If the optimal solution is unattainable then it is reasonable to sacrifice optimality and settle for a "good" feasible solution that can be computed efficiently. Of course, we would like to sacrifice as little optimality as possible, while gaining as much as possible in efficiency. Trading-off optimality in favor of tractability is the paradigm of approximation algorithms.

The main themes of this book revolve around the design of such algorithms and the "closeness" to the optimum that is achievable in polynomial time. To evaluate the limits of approximability, it is important to derive lower bounds or inapproximability results. In some cases, approximation algorithms must satisfy additional structural requirements such as being on-line, or working within limited space. This book reviews the design techniques for such algorithms and the developments in this area since its inception about three decades ago.

WHAT CAN APPROXIMATION ALGORITHMS DO FOR YOU: AN ILLUSTRATIVE EXAMPLE

0.1

Consider the following problem that motivates the study of approximation algorithms and also happens to be the first ever treated in the approximation algorithms framework.

Picture yourself at 9 a.m., on the shop floor, facing 8 machines that will be ready for work at 10 a.m., and 147 jobs of various durations waiting to be processed. There is a 9 p.m. basketball game on TV which you would hate to miss, but you have to stay until all the jobs are finished. You would want to assign jobs to machines so that you can get home as early as possible.

However, there is one catch: the problem is an instance of the well-known *minimum makespan* problem which is *NP*-hard. Being *NP*-hard means not only that there is no known efficient algorithm for solving the problem, but also that it is quite unlikely that one exists.

Given that this optimization problem is *NP*-hard, what is the next step? For years *NP*-hard problems were treated with integer programming tools or "heuristics." Integer programming tools are forms of implicit enumeration algorithms that combine efficient derivations of lower and upper bounds with a hopeful search for an optimal solution. The amount of time required to solve even a typical moderate-size optimization problem is exorbitant. Instead, the user interrupts the enumeration process either when the current solution is deemed satisfactory, or when the running time has exceeded reasonable limit. The point is that integer programming algorithms provide no *guarantee*. It is impossible to tell if 5 additional minutes of running time would get you a significantly better solution, or if 5 more days of running time would yield no improvement at all.

Besides the issues of guarantee of quality of solution and reasonable running time, the classic integer programming tool, *Branch-and-Bound*, needs good bounds which are feasible solutions as well as good estimates of the value of the optimum (lower bounds for minimization problems—upper bounds for maximization). Approximation algorithms address both the issue of guarantee and of making good feasible solutions available. The analysis of approximation algorithms always involves deriving estimates on the value of the optimum. As such, approximation algorithms and their analysis are useful in traditional integer programming techniques.

... back on the "floor" the time is a few minutes before 10:00 a.m. and it is too late to start running a branch-and-bound procedure since you may end up worse off compared to a solution that you can quickly guess. Indeed how about using some rules of thumb to guess a reasonable solution? Say, place any job on any machine as soon as the machine becomes available. That sounds sensible, but a quick calculation shows that the machine that finishes last will do so at 10 p.m. It would certainly help to get a clue as to how much better an optimal schedule value can be, even if the schedule itself is unattainable. In essence we would like to know how good is the solution delivered by the above rule compared to the optimum. Indeed, Graham proved in 1966 [Gra66] that this rule gives a solution with relative error no more than 100%. In other words, while the rule delivers a schedule lasting 12 hours, the optimum could be as short as 6 hours, but no less. At this point you are assured that no solution will get you off before 4 p.m.

Heuristics is the nomenclature for rules-of-thumb which in itself is a euphemism for simple and invariably polynomial algorithms. Heuristics work quickly and efficiently. The quality of the solution they deliver is another matter altogether. Prior to the advent of the approximation algorithms' method of analysis, the performance of a heuristic was judged by running it on a benchmark set of problem instances and comparing it with the performance of other heuristics on the same benchmark.

The obvious setbacks to such an approach were already evident in the pre-complexity days. The benchmark set is typically a small sample which is not necessarily a good representative of general problem instances. In addition, improvements in the quality of a solution as measured by the benchmark set are not necessarily a good predictor of possible improvements in general cases. There was clearly a need for well-defined analysis to evaluate the quality of heuristics. When such an analysis emerged it had the added benefit of enhancing our understanding of the problem and providing insight as

to what makes a solution "right." This insight made it possible to improve the heuristics and the integer programming algorithms. On the other hand, many heuristics with excellent empirical performance have so far eluded formal analysis.

Garey, Graham, and Ullman [GGU72] and later Johnson [Joh74] formalized the concept of an *approximation algorithm*. An approximation algorithm is necessarily polynomial, and is evaluated by the worst case possible relative error over all possible instances of the problem. An algorithm $\mathcal{A}$ is said to be a δ-approximation algorithm for a minimization problem P if for every instance I of P it delivers a solution that is at most δ times the optimum. Naturally, $\delta > 1$ and the closer it is to 1, the better. Similarly, for maximization problems a δ-approximation algorithm delivers for every instance I a solution that is at least δ times the optimum. In that case $\delta < 1$. δ is referred to as the *approximation ratio*, or *performance guarantee*, or *worst case ratio*, or *worst case error bound*, or *approximation factor*. For the maximization problem it is also common to refer to $1/\delta$ as the approximation factor.

As stated above, the sensible rule described for the minimum makespan problem gives a solution with relative error no more than 100%. As such, it is a 2-approximation algorithm. (Details on the analysis of this approximation algorithm, called List Scheduling algorithm, are given in Chapter 1.) Graham further showed that a different heuristic that assigns the longest remaining job to the first available machine gives a better worst case error ratio of 4/3. With this slightly modified rule you get a 10-hour schedule that ends at 8 p.m. Furthermore, now you know that the best solution—the optimum schedule—will last 7 and a 1/2 hours and thus will not end before 5:30 p.m.

For the minimum makespan problem, considerably better approximation algorithms have been devised by Hochbaum and Shmoys [HS87]. They described a family of approximation algorithms for the minimum makespan problem so that algorithm $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm (Section 9.3.2 contains a description of this family of algorithms). This means you can get an error as small as you like, but have to pay significantly in terms of increased running time. The rate of increase in running time is a known function of ϵ, so one can decide on the appropriate trade-off for the situation. For instance, an algorithm that guarantees a ratio no more than 6/5 works in several hundred steps (more precisely, in $O(n \log n)$ steps for n jobs) and hence can deliver a solution in a fraction of a second with a guarantee of no more than 20% relative error (this specific algorithm is described in [HS87]). Applying that 6/5-approximation algorithm gives you an assignment of the 147 jobs that is 9 hours long, terminates at 7 p.m. and guarantees that the optimum is no earlier than 5:30 p.m. Considering it satisfactory you can use this schedule, or, if there is still time till 10 a.m., run one of the better approximation algorithms in the family to see if it results in yet a better schedule. Even if not, we at least end up with a higher estimate for the finish time of the optimum, thus reducing "regret."

This trade-off family of algorithms is called an approximation scheme. While such success is not typical for other *NP*-hard problem it demonstrates that the detailed analysis and insight gained lead to the generation of this tool of approximation scheme for the makepsan problem. With the increased interest in the area of approximation algorithms, more and more empirically successful heuristics for other problems are being analyzed and theoretically understood. This results in the ability to derive closer to optimum solutions for these problems than previously done.

Having introduced informally the main concepts, we now proceed to a more systematic introduction of the basic definitions.

0.2 FUNDAMENTALS AND CONCEPTS

Foremost among the concepts is that of a δ-approximation algorithm. An approximation algorithm is always assumed to be "efficient" or more precisely, polynomial. We also assume that the approximation algorithm delivers a feasible solution to some *NP*-hard problem that has a set of instances $\{I\}$.

DEFINITION 0.1 A polynomial algorithm, $\mathcal{A}$, is said to be a δ-approximation algorithm if for every problem instance I with an optimal solution value $OPT(I)$,

$$OPT(I) \leq \delta.$$

As mentioned before, $\delta \geq 1$ for minimization problems and ≤ 1 for maximization problems. The smallest value of δ is the approximation (or performance) ratio $R_{\mathcal{A}}$ of the algorithm $\mathcal{A}$.

The value of δ is referred to by any of the following terms or their variations

- Worst case bound
- Worst case performance
- Approximation factor
- Approximation ratio
- Performance bound
- Performance ratio
- Error ratio

and several others.

For maximization problems, sometimes $\frac{1}{\delta}$ is considered to be the approxmiation ratio/factor.

Unless otherwise specified, we always mean for δ to be the *absolute performance ratio*. However, in some cases the error involves an additive term. In those cases the notion of asymptotic performance ratio is relevant.

DEFINITION 0.2 The *absolute performance ratio*, $R_{\mathcal{A}}$, of an approximation algorithm $\mathcal{A}$ is,

$$R_{\mathcal{A}} = \inf\{r \geq 1 | R_{\mathcal{A}}(I) \leq r \text{ for all problem instances } I\}.$$

and the *asymptotic performance ratio* $R_{\mathcal{A}}^{\infty}$ for $\mathcal{A}$ is,

$$R_{\mathcal{A}}^{\infty} = \inf\{r \geq 1 | \exists n \in Z^{+}, R_{\mathcal{A}}(I) \leq r \ \forall I \text{ s.t. } I \geq n\}.$$

In Chapter 9 we illustrate examples where the difference between these two ratios is significant. For online algorithms there is an analogous concept of *competitive ratio* (asymptotic). There, however, the input instance is a sequence and the comparison is to the performance of an optimal offline algorithm on the same sequence, rather than to an

optimal solution value. Chapter 12 provides details on the motivation and definition of this concept.

The concept of $R_{\mathcal{A}}$ is a *worst case* notion meaning that it suffices to have a single "bad" instance to render the value of δ larger than it is for all other encountered instances. Typically, the observed performance of an approximation algorithm, as reflected in the gap between the optimal solution and the delivered solution, is considerably better than would be indicated by the performance ratio. This has been evident in every experimental study on specific problem instances. Ideally we would like to be able to predict the actual performance of the algorithm.

One way of addressing this concern is through the use of *average case analysis*. In average case analysis we assume knowledge of the distribution of the problem's instances. This knowledge permits a tightening of the assessment of the optimal value as well as allows for tailor-fitting algorithms to the features of the distribution. Average case analysis is illustrated in Chapter 2 of the book, primarily for the bin packing problem.

The efficiency of an approximation algorithm is another important issue. While we assume that every approximation algorithm is polynomial, there is a vast variety of polynomial algorithms some of which are decidedly inefficient and impractical. Within the range of "practical" polynomiality we may want to invest more running time in order to get a better approximation bound. Such a trade-off comes in several versions of *approximation schemes*.

DEFINITION 0.3 A family of approximation algorithms for a problem $\mathcal{P}$, $\{\mathcal{A}_\epsilon\}_\epsilon$, is called a *polynomial approximation scheme* or PAS, if algorithm $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm and its running time is polynomial in the size of the input for a fixed ϵ.

DEFINITION 0.4 A family of approximation algorithms for a problem $\mathcal{P}$, $\{\mathcal{A}_\epsilon\}_\epsilon$, is called a *fully polynomial approximation scheme* or FPAS, if algorithm $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm and its running time is polynomial in the size of the input **and** $1/\epsilon$.

When a FPAS is a family of *randomized* algorithms it will be called *fully polynomial randomized approximation scheme* or FPRAS. In that case the approximation is guaranteed with probability that is large enough (e.g. 3/4 as in Chapter 12). Chapter 9 discusses examples of PAS and FPAS, and Chapter 12 discusses examples of FPRAS. Chapter 10 reviews the class of Max-$\mathcal{SNP}$-hard problems that cannot have PAS, unless $NP = P$.

Lower bounds on approximability are a major concern as we always want to know whether a better approximation exists which we simply failed to identify, or if better approximations are impossible. For a large number of important problems there have recently been a slew of discouraging results in the sense that their approximability limits are quite bad. Among these problems the maximum clique is prominent. In the maximum clique problem the aim is to find the largest number of vertices in a graph all of which are linked with each other via edges. A trivial solution, and not a very good one, is to take a single vertex as the clique. The inapproximability results demonstrate that unless $NP = P$ we cannot guarantee to do substantially better. This problem, and the recent techniques for proving lower bounds, are the main topics of Chapter 10. More traditional techniques that prove lower bounds on approximability are described in Chapter 9.

In a number of practical situations an algorithm is required to have a specific structure. For instance, when the data instance is not available *a-priori*, then the decision, or the algorithm, has to be executed *on-line* with only partial information. This is, for example, the case with the minimum makespan problem when jobs arrive continually and the machines have to be assigned and running without waiting for the stream of jobs to end. Graham's 2-approximation algorithm previously described offers the extra bonus of also being on-line: a job in the pool is assigned to the first available machine. When we want to assess the performance of an on-line algorithm the notion of worst case ratio is inappropriate as it would be meaningless to compare the solution delivered by such algorithm with optimal value that does not depend on the arrival sequence of the jobs. Instead we compare it with an optimal algorithm that is off-line, meaning that it has the *a-priori* information about the sequence of the arriving jobs. For this purpose the concept of *competitive ratio* and related concepts are introduced (see Chapter 13) for analyzing on-line problems. These concepts measure the effects of partial information.

Other types of approximation algorithms are required to work in *restricted space* or in dynamic fashion. Such algorithms, as well as several other interesting variants, are described in Chapter 2.

Randomized algorithms were found to be extremely useful in general algorithm design (see the recent book by Motwani and Raghavan [MR95]). These algorithms make random choices during execution. In the context of approximation algorithms, the randomized approach had a dramatic affect in introducing novel approaches. Randomization is generally combined with the continuous techniques of linear programming and semidefinite programming. Randomized algorithms are also the only algorithms known that provide any sort of estimate for solutions to *counting* problems. Problems, such as counting the number of perfect matchings in a graph or assessing the volume of a polytope are harder than the corresponding optimization problems. For the few instances when solutions are known (Chapter 12) they are achieved via randomized techniques.

Finally, the continuous optimization problems of *linear programming* and *semidefinite programming* are of prime importance in approximations. Since it is possible to formulate optimization problems as integer programming, linear programming relaxations, (in which the requirement of integrality is omitted), provide a bound (lower bound for a minimization problem, upper bound for maximization). If a semidefinite programming relaxation of the problem is known, then the bound is frequently tighter (closer to the actual optimum) than that obtained by linear programming. It is likely that other, more general, nonlinear relaxations can be tighter still, but for that to be useable, those nonlinear problems need to be solvable in polynomial time. This is promising to be the direction of future developments in approximations.

The reader will need to have some background in linear programming, as pertains to duality theory, and the knowledge that the problem is polynomial. A good introduction to linear programming is given in a recent book by Saigal [S95]. For semidefinite programming the relevant references are mentioned in Chapter 11.

0.3 OBJECTIVES AND ORGANIZATION OF THIS BOOK

This book is aimed at practitioners interested in specific application areas, as well as the computer science and operations research community interested in design tools for algorithms in general and approximation algorithms in particular. Our goal here is to layout the variety of approaches and techniques that are typical and most effective for approximations.

The variety and versatility make it difficult to navigate in this area. One goal of this book is to introduce a framework, review of applications and unifying techniques in the analysis of approximation algorithms. Such unifying features have only emerged recently as the area has matured.

The chapters in this book are written so as to be self-contained to the greatest extent. Each chapter has its own list of references that provide a perspective on the work done in the particular subarea. The reader is assumed to have prior background in the design and complexity of algorithms, as well as in linear programming and stochastic processes. Each chapter gives references to appropriate tutorial material. There are exercises scattered throughout the text that highlight important extensions or ask the reader to verify some of the claims.

The chapters of the book can be read in any order. The reader can identify specific techniques or problems of interest in the index and in the glossary of problems that include pointers to chapters where the problems are discussed. The glossary is arranged in an alphabetical order relating to some particular key words in the problem title. The glossary, however, is not organized by topics and it may be necessary to use the index in order to pinpoint a specific problem definition.

The first three chapters follow the chronological developments in the field. We chose to have the topic of scheduling as the opening chapter since scheduling problems were the first historically to be analyzed for approximations. The area of scheduling has remained very active and novel techniques have been devised and used. Chapter 1, indeed, reviews the early work, but is mostly devoted to the discussion of recent developments. Another problem that was extensively analyzed in the early days of approximation algorithms analysis is the *bin packing* problem. Chapter 2 focuses on this problem and its extensions. It is the only chapter to cover average case analysis, for which bin packing has been a showcase problem. The chapter also reviews on-line algorithms that are space restricted and *dynamic* algorithms and *open* versus *closed* on-line algorithms.

The *set cover* problem has been considered one of the most prominent problems in optimization. The first algorithm to use linear programming and duality was devised for this problem and its special case—the vertex cover problem. Not surprisingly, this algorithm came shortly after the discovery that linear programming is a polynomial problem.While the performance of the first algorithms was not improved since the early 1980s, there has been substantial progress on specially structured set cover problem, as well as for the vertex cover problem and it complement—the independent set problem. Chapter 3 reviews these algorithms and provides an up-to-date summary of best approximations. It includes an analysis that explains the particular usefulness of linear programming relaxation for the vertex cover and independent set and other integer programs with

two variables per inequality. The chapter also reviews the important greedy algorithm and its properties.

Chapter 4 presents refinements of the linear programming based technique described in Chapter 3. It describes a number of problems in the context of network design applications, that can be posed as covering problems. It then shows how the technique using primal and dual solutions leads to the design of stronger approximation algorithms than does general covering for a surprisingly vast collection of problems.

Chapter 5 is concerned with problems that involve partitioning of graphs such as separation or cut problems. The techniques here, again, are dominated by the use of linear programming relaxations and their duals. Several randomized algorithms are reviewed as well. The chapter addresses problems that make use of separation, or divide-and-conquer algorithms for problems such as the multicommodity and the linear arrangement.

Chapters 6 is devoted to connectivity problems. While these problems can be thought of as network design problems, the focus here is on graph algorithmic techniques. Chapter 7 uses graph algorithms to solve problems of network design characterized by the objective of minimizing the maximum degree node in the required structure (a recurring scenario in limited capacity setups). One of the most dramatic successes of approximation algorithms is described in this chapter—the derivation of a solution which is within one unit of the optimum for the problem of finding a spanning tree for which the maximum vertex degree is minimized. This problem was previously the target of a variety of heavy duty machinery, none of which approached the quality of the solution and the efficiency of the approximation described in this chapter.

Chapter 8 is devoted to the algorithms and techniques that are most successful when the problems are given in the plane or in a Euclidean space. These problems include the Traveling Salesman problem (TSP), the Steiner tree problem, triangulations, and a variety of clustering problems. Approximation algorithms for these types of problems use different techniques, compared to those used for non-Euclidean problems, that rely on the field of computational geometry. The TSP is not covered comprehensively in this book, as it has been extensively covered in the literature to date. The bottleneck version of TSP is analyzed in the next chapter.

Chapter 9 addresses problems that are not otherwise covered in the book, with emphasis on the "quality" of the approximation bound and on how much it can be improved. It includes samples of constant approximation algorithms: PASs, FPASs; approximation algorithms that are provably best (unless $NP = P$); some lower bound results; illustrations of the differences between absolute and asymptotic worst case ratios and approximation algorithms that are within one unit of the optimum. The chapter discusses problems with a wide scope of applications including location, clustering network communication, covering and packing with certain objects, scheduling, and more.

The recent lower bound techniques based on probabilistically checkable proofs are reviewed in Chapter 10. This chapter is concerned with the limits on approximability of problems rather than with the design of approximation algorithms.

In Chapter 11 there are randomized algorithms analyzed in conjunction with linear programming and semidefinite programming. (It is interesting that these two continuous techniques work so well together with the technique of randomization that has been proven successful for algorithm design in general.) The randomization makes use of the fractional values and interprets them as probabilities for rounding up the given variable.

As was the case for the use of linear programming in approximations, the discovery that semidefinite programming is useful for approximations came shortly after it was established that semidefinite programming is a polynomial problem. The chapter also describes the important topic of *derandomization*—the conversion of a randomized algorithm to a deterministic one.

Chapter 12 is about Markov chain techniques that have proven useful for providing an approximate answer to various counting problems. The idea is to use sampling in an efficient way in order to estimate the magnitude of the answer from a polynomially restricted search.

Chapter 13 describes extensively online (on-line) algorithms and their major applications. It reviews the various design techniques and recent results on lower and upper bounds for the paging, k-server, metrical task system, and other online problems.

0.4 ACKNOWLEDGMENTS

I would like to thank all the contributors for their expertise and enthusiasm for this project. In particular, I benefited from David Shmoys' editorial help, from Marshall Bern's hawk-eye for fonts and design details, and from Samir Khuller and Sanjeev Arora's helpful feedback. Through the arduous months (that extended to years) that it took to complete the book, I enjoyed the help and encouragement of my students, Anu Pathria, Youxun Shen, and Eli Olinick. The numerous conversations I had with Anu formed my views on the orientation of the book and the presentation of the material.

Along the way I had substantial help from my friends and colleagues. I would like to note in particular Ilan Adler, who always helped me put things in perspective, Olivier Goldschmidt, and Seffi Naor. Cliff Stein was most helpful in providing information and pointers on the superstring problem.

The technical production of the book turned out to be a mammoth task for which I needed all the help I could get. I thank Mandy Simondson for her supportive response to emergencies, and to William Baxter, the LaTeX wiz, from whom I learned more than I ever cared to know about LaTeX, LaTeX 2_ε, and their intricacies. I wish to acknowledge the continuing support from ONR of my research on algorithms and approximation algorithms. Finally, to Adriana for her warm friendship and inspiring paintings, to Aharon, Allon, and Daniel whose enthusiasm for seeing the Hochbaum name on a book cover helped compensate for extended work days, and to Belle and Ivan for always being there for us.

Dorit S. Hochbaum

REFERENCES

[GGU72] M. R. Garey, R.L. Graham and J. D. Ullman. Worst case analysis of memory allocation algorithms. in Proc. of the 4th ACM Symp.on Theory of Computing. 143-150, 1972.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman, San Francisco, 1979.

[Gra66] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J. 45,* 1563–1581, 1966.

[HS87] D. S. Hochbaum, D. B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of ACM* 34:1, 144–162, 1987.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9, 256-278, 1974.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, Cambridge, 1995.

[S95] R. Saigal. *Linear programming: a modern integrated analysis.* Kluwer Academic Publishers, Boston, 1995.

CONTENTS