Vijay V. Vazirani

College of Computing
Georgia Institute of Technology

Approximation Algorithms

Springer

Berlin Heidelberg New York
Barcelona Hong Kong
London Milan Paris
Singapore Tokyo

To my parents

Preface

Although this may seem a paradox, all exact science is dominated by the idea of approximation.
Bertrand Russell (1872–1970)

Most natural optimization problems, including those arising in important application areas, are **NP**-hard. Therefore, under the widely believed conjecture that $\mathbf{P} \neq \mathbf{NP}$, their exact solution is prohibitively time consuming. Charting the landscape of approximability of these problems, via polynomial time algorithms, therefore becomes a compelling subject of scientific inquiry in computer science and mathematics. This book presents the theory of approximation algorithms as it stands today. It is reasonable to expect the picture to change with time.

The book is divided into three parts. In Part I we cover a combinatorial algorithms for a number of important problems, using a wide variety of algorithm design techniques. The latter may give Part I a non-cohesive appearance. However, this is to be expected – nature is very rich, and we cannot expect a few tricks to help solve the diverse collection of **NP**-hard problems. Indeed, in this part, we have purposely refrained from tightly categorizing algorithmic techniques so as not to trivialize matters. Instead, we have attempted to capture, as accurately as possible, the individual character of each problem, and point out connections between problems and algorithms for solving them.

In Part II, we present linear programming based algorithms. These are categorized under two fundamental techniques: rounding and the primal–dual schema. But once again, the exact approximation guarantee obtainable depends on the specific LP-relaxation used, and there is no fixed recipe for discovering good relaxations, just as there is no fixed recipe for proving a theorem in mathematics (readers familiar with complexity theory will recognize this as the philosophical point behind the $\mathbf{P} \neq \mathbf{NP}$ question).

Part III covers four important topics. The first is the problem of finding a shortest vector in a lattice which, for several reasons, deserves individual treatment (see Chapter 27).

The second topic is the approximability of counting, as opposed to optimization, problems (counting the number of solutions to a given instance). The counting versions of all known **NP**-complete problems are #**P**-complete[1]. Interestingly enough, other than a handful of exceptions, this is true of problems in **P** as well. An impressive theory has been built for ob-

[1] However, there is no theorem to this effect yet.

taining efficient approximate counting algorithms for this latter class of problems. Most of these algorithms are based on the Markov chain Monte Carlo (MCMC) method, a topic that deserves a book by itself and is therefore not treated here. In Chapter 28 we present combinatorial algorithms, not using the MCMC method, for two fundamental counting problems.

The third topic is centered around recent breakthrough results, establishing hardness of approximation for many key problems, and giving new legitimacy to approximation algorithms as a deep theory. An overview of these results is presented in Chapter 29, assuming the main technical theorem, the PCP Theorem. The latter theorem, unfortunately, does not have a simple proof at present.

The fourth topic consists of the numerous open problems of this young field. The list presented should by no means be considered exhaustive, and is moreover centered around problems and issues currently in vogue. Exact algorithms have been studied intensively for over four decades, and yet basic insights are still being obtained. Considering the fact that among natural computational problems, polynomial time solvability is the exception rather than the rule, it is only reasonable to expect the theory of approximation algorithms to grow considerably over the years.

The set cover problem occupies a special place, not only in the theory of approximation algorithms, but also in this book. It offers a particularly simple setting for introducing key concepts as well as some of the basic algorithm design techniques of Part I and Part II. In order to give a complete treatment for this central problem, in Part III we give a hardness result for it, even though the proof is quite elaborate. The hardness result essentially matches the guarantee of the best algorithm known – this being another reason for presenting this rather difficult proof.

Our philosophy on the design and exposition of algorithms is nicely illustrated by the following analogy with an aspect of Michelangelo's art. A major part of his effort involved looking for interesting pieces of stone in the quarry and staring at them for long hours to determine the form they naturally wanted to take. The chisel work exposed, in a minimalistic manner, this form. By analogy, we would like to start with a clean, simply stated problem (perhaps a simplified version of the problem we actually want to solve in practice). Most of the algorithm design effort actually goes into understanding the algorithmically relevant combinatorial structure of the problem. The algorithm exploits this structure in a minimalistic manner. The exposition of algorithms in this book will also follow this analogy, with emphasis on stating the structure offered by problems, and keeping the algorithms minimalistic.

An attempt has been made to keep individual chapters short and simple, often presenting only the key result. Generalizations and related results are relegated to exercises. The exercises also cover other important results which could not be covered in detail due to logistic constraints. Hints have been provided for some of the exercises; however, there is no correlation between the degree of difficulty of an exercise and whether a hint is provided for it.

This book is suitable for use in advanced undergraduate and graduate level courses on approximation algorithms. It has more than twice the material that can be covered in a semester long course, thereby leaving plenty of room for an instructor to choose topics. An undergraduate course in algorithms and the theory of **NP**-completeness should suffice as a prerequisite for most of the chapters. For completeness, we have provided background information on several topics: complexity theory in Appendix A, probability theory in Appendix B, linear programming in Chapter 12, semidefinite programming in Chapter 26, and lattices in Chapter 27. (A disproportionate amount of space has been devoted to the notion of self-reducibility in Appendix A because this notion has been quite sparsely treated in other sources.) This book can also be used is as supplementary text in basic undergraduate and graduate algorithms courses. The first few chapters of Part I and Part II are suitable for this purpose. The ordering of chapters in both these parts is roughly by increasing difficulty.

In anticipation of this wide audience, we decided not to publish this book in any of Springer's series – even its prestigious Yellow Series. (However, we could not resist spattering a patch of yellow on the cover!) The following translations are currently planned: French by Claire Kenyon, Japanese by Takao Asano, and Romanian by Ion Măndoiu. Corrections and comments from readers are welcome. We have set up a special email address for this purpose: approx@cc.gatech.edu.

Finally, a word about practical impact. With practitioners looking for high performance algorithms having error within 2% or 5% of the optimal, what good are algorithms that come within a factor of 2, or even worse, $O(\log n)$, of the optimal? Further, by this token, what is the usefulness of improving the approximation guarantee from, say, factor 2 to 3/2?

Let us address both issues and point out some fallacies in these assertions. The approximation guarantee only reflects the performance of the algorithm on the most pathological instances. Perhaps it is more appropriate to view the approximation guarantee as a measure that forces us to explore deeper into the combinatorial structure of the problem and discover more powerful tools for exploiting this structure. It has been observed that the difficulty of constructing tight examples increases considerably as one obtains algorithms with better guarantees. Indeed, for some recent algorithms, obtaining a tight example has been a paper by itself (e.g., see Section 26.7). Experiments have confirmed that these and other sophisticated algorithms do have error bounds of the desired magnitude, 2% to 5%, on typical instances, even though their worst case error bounds are much higher. Additionally, the theoretically proven algorithm should be viewed as a core algorithmic idea that needs to be fine tuned to the types of instances arising in specific applications.

We hope that this book will serve as a catalyst in helping this theory grow and have practical impact.

Acknowledgments

This book is based on courses taught at the Indian Institute of Technology, Delhi in Spring 1992 and Spring 1993, at Georgia Tech in Spring 1997, Spring 1999, and Spring 2000, and at DIMACS in Fall 1998. The Spring 1992 course resulted in the first set of class notes on this topic. It is interesting to note that more than half of this book is based on subsequent research results.

Numerous friends – and family members – have helped make this book a reality. First, I would like to thank Naveen Garg, Kamal Jain, Ion Măndoiu, Sridhar Rajagopalan, Huzur Saran, and Mihalis Yannakakis – my extensive collaborations with them helped shape many of the ideas presented in this book. I was fortunate to get Ion Măndoiu's help and advice on numerous matters – his elegant eye for layout and figures helped shape the presentation. A special thanks, Ion!

I would like to express my gratitude to numerous experts in the field for generous help on tasks ranging all the way from deciding the contents and its organization, providing feedback on the writeup, ensuring correctness and completeness of references to designing exercises and helping list open problems. Thanks to Sanjeev Arora, Alan Frieze, Naveen Garg, Michel Goemans, Mark Jerrum, Claire Kenyon, Samir Khuller, Daniele Micciancio, Yuval Rabani, Sridhar Rajagopalan, Dana Randall, Tim Roughgarden, Amin Saberi, Leonard Schulman, Amin Shokrollahi, and Mihalis Yannakakis, with special thanks to Kamal Jain, Éva Tardos, and Luca Trevisan.

Numerous other people helped with valuable comments and discussions. In particular, I would like to thank Sarmad Abbasi, Cristina Bazgan, Rogerio Brito Gruia Calinescu, Amit Chakrabarti, Mosses Charikar, Joseph Cheriyan, Vasek Chvátal, Uri Feige, Cristina Fernandes, Ashish Goel, Parikshit Gopalan, Mike Grigoriadis, Sudipto Guha, Dorit Hochbaum, Howard Karloff, Leonid Khachian, Stavros Kolliopoulos, Jan van Leeuwen, Nati Lenial, George Leuker, Vangelis Markakis, Aranyak Mehta, Rajeev Motwani, Prabhakar Raghavan, Satish Rao, Miklos Santha, Jiri Sgall, David Shmoys, Alistair Sinclair, Prasad Tetali, Pete Veinott, Ramarathnam Venkatesan, Nisheeth Vishnoi, and David Williamson. I am sure I am missing several names – my apologies and thanks to these people as well. A special role was played by the numerous students who took my courses on this topic and scribed notes. It will be impossible to individually remember their names. I would like to express my gratitude collectively to them.

I would like to thank IIT Delhi – with special thanks to Shachin Maheshwari – Georgia Tech, and DIMACS for providing pleasant, supportive and academically rich environments. Thanks to NSF for support under grants CCR-9627308 and CCR-9820896.

It was a pleasure to work with Hans Wössner on editorial matters. The personal care with which he handled all such matters and his sensitivity to an author's unique point of view were especially impressive. Thanks also to Frank Holzwarth for sharing his expertise with LaTeX.

A project of this magnitude would be hard to pull off without wholehearted support from family members. Fortunately, in my case, some of them are also fellow researchers – my wife, Milena Mihail, and my brother, Umesh Vazirani. Little Michel's arrival, halfway through this project, brought new joys and energies, though made the end even more challenging! Above all, I would like to thank my parents for their unwavering support and inspiration – my father, a distinguished author of several Civil Engineering books, and my mother, with her deep understanding of Indian Classical Music. This book is dedicated to them.

Atlanta, Georgia, May 2001 Vijay Vazirani

Table of Contents

Part II. LP-Based Algorithms

1 Introduction

NP-hard optimization problems exhibit a rich set of possibilities, all the way from allowing approximability to any required degree, to essentially not allowing approximability at all. Despite this diversity, underlying the process of design of approximation algorithms are some common principles. We will explore these in the current chapter.

An optimization problem is polynomial time solvable only if it has the algorithmically relevant combinatorial structure that can be used as "footholds" to efficiently home in on an optimal solution. The process of designing an exact polynomial time algorithm is a two-pronged attack: unraveling this structure in the problem and finding algorithmic techniques that can exploit this structure.

Although **NP**-hard optimization problems do not offer footholds for finding optimal solutions efficiently, they may still offer footholds for finding near-optimal solutions efficiently. So, at a high level, the process of design of approximation algorithms is not very different from that of design of exact algorithms. It still involves unraveling the relevant structure and finding algorithmic techniques to exploit it. Typically, the structure turns out to be more elaborate, and often the algorithmic techniques result from generalizing and extending some of the powerful algorithmic tools developed in the study of exact algorithms.

On the other hand, looking at the process of designing approximation algorithms a little more closely, one can see that it has its own general principles. We illustrate some of these principles in Section 1.1, using the following simple setting.

Problem 1.1 (Vertex cover) Given an undirected graph $G = (V, E)$, and a cost function on vertices $c : V \rightarrow \mathbf{Q}^+$, find a minimum cost *vertex cover*, i.e., a set $V' \subseteq V$ such that every edge has at least one endpoint incident at V'. The special case, in which all vertices are of unit cost, will be called the *cardinality vertex cover problem*.

Since the design of an approximation algorithm involves delicately attacking **NP**-hardness and salvaging from it an efficient approximate solution, it will be useful for the reader to review some key concepts from complexity theory. Appendix A and some exercises in Section 1.3 have been provided for this purpose.

It is important to give precise definitions of an **NP**-optimization problem and an approximation algorithm for it (e.g., see Exercises 1.9 and 1.10). Since these definitions are quite technical, we have moved them to Appendix A. We provide essentials below to quickly get started.

An **NP**-optimization problem Π is either a minimization or a maximization problem. Each valid instance I of Π comes with a nonempty set of feasible solutions, each of which is assigned a nonnegative rational number called its objective function value. There exist polynomial time algorithms for determining validity, feasibility, and the objective function value. A feasible solution that achieves the optimal objective function value is called an optimal solution. $\mathrm{OPT}_\Pi(I)$ will denote the objective function value of an optimal solution to instance I. We will shorten this to OPT when there is no ambiguity. For the problems studied in this book, computing $\mathrm{OPT}_\Pi(I)$ is **NP**-hard.

For example, valid instances of the vertex cover problem consist of an undirected graph $G = (V, E)$ and a cost function on vertices. A feasible solution is a set $S \subseteq V$ that is a cover for G. Its objective function value is the sum of costs of all vertices in S. A minimum cost such set is an optimal solution.

An approximation algorithm, $\mathcal{A}$, for Π produces, in polynomial time, a feasible solution whose objective function value is "close" to the optimal; by "close" we mean within a guaranteed factor of the optimal. In the next section, we will present a factor 2 approximation algorithm for the cardinality vertex cover problem, i.e., an algorithm that finds a cover of cost $\leq 2 \cdot \mathrm{OPT}$ in time polynomial in $|V|$.

1.1 Lower bounding OPT

When designing an approximation algorithm for an **NP**-hard **NP**-optimization problem, one is immediately faced with the following dilemma. In order to establish the approximation guarantee, the cost of the solution produced by the algorithm needs to be compared with the cost of an optimal solution. However, for such problems, not only is it **NP**-hard to find an optimal solution, but it is also **NP**-hard to compute the cost of an optimal solution (see Appendix A). In fact, in Section A.5 we show that computing the cost of an optimal solution (or even solving its decision version) is precisely the difficult core of such problems. So, how do we establish the approximation guarantee? Interestingly enough, the answer to this question provides a key step in the design of approximation algorithms.

Let us demonstrate this in the context of the cardinality vertex cover problem. We will get around the difficulty mentioned above by coming up with a "good" polynomial time computable *lower bound* on the size of the optimal cover.

1.1.1 An approximation algorithm for cardinality vertex cover

We provide some definitions first. Given a graph $H = (U, F)$, a subset of the edges $M \subseteq F$ is said to be a *matching* if no two edges of M share an endpoint. A matching of maximum cardinality in H is called a *maximum matching*, and a matching that is maximal under inclusion is called a *maximal matching*. A maximal matching can clearly be computed in polynomial time by simply greedily picking edges and removing endpoints of picked edges. More sophisticated means lead to polynomial time algorithms for finding a maximum matching as well.

Let us observe that the size of a maximal matching in G provides a lower bound. This is so because *any* vertex cover has to pick at least one endpoint of each matched edge. This lower bounding scheme immediately suggests the following simple algorithm:

Algorithm 1.2 (Cardinality vertex cover)

Find a maximal matching in G and output the set of matched vertices.

Theorem 1.3 *Algorithm 1.2 is a factor 2 approximation algorithm for the cardinality vertex cover problem.*

Proof: No edge can be left uncovered by the set of vertices picked – otherwise such an edge could have been added to the matching, contradicting its maximality. Let M be the matching picked. As argued above, $|M| \leq \text{OPT}$. The approximation factor follows from the observation that the cover picked by the algorithm has cardinality $2\,|M|$, which is at most $2 \cdot \text{OPT}$. $\square$

Observe that the approximation algorithm for vertex cover was very much related to, and followed naturally from, the lower bounding scheme. This is in fact typical in the design of approximation algorithms. In Part II of this book, we show how linear programming provides a unified way of obtaining lower bounds for several fundamental problems. The algorithm itself is designed around the LP that provides the lower bound.

1.1.2 Can the approximation guarantee be improved?

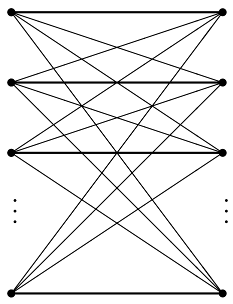
The following questions arise in the context of improving the approximation guarantee for cardinality vertex cover:

1. Can the approximation guarantee of Algorithm 1.2 be improved by a better analysis?

2. Can an approximation algorithm with a better guarantee be designed using the lower bounding scheme of Algorithm 1.2, i.e., size of a maximal matching in G?
3. Is there some other lower bounding method that can lead to an improved approximation guarantee for vertex cover?

Example 1.4 shows that the answer to the first question is "no", i.e., the analysis presented above for Algorithm 1.2 is tight. It gives an infinite family of instances in which the solution produced by Algorithm 1.2 is twice the optimal. An infinite family of instances of this kind, showing that the analysis of an approximation algorithm is tight, will be referred to as a *tight example*. The importance of finding tight examples for an approximation algorithm one has designed cannot be overemphasized. They give critical insight into the functioning of the algorithm and have often led to ideas for obtaining algorithms with improved guarantees. (The reader is advised to run algorithms on the tight examples presented in this book.)

Example 1.4 Consider the infinite family of instances given by the complete bipartite graphs $K_{n,n}$.



When run on $K_{n,n}$, Algorithm 1.2 will pick all $2n$ vertices, whereas picking one side of the bipartition gives a cover of size n. □

Let us assume that we will establish the approximation factor for an algorithm by simply comparing the cost of the solution it finds with the lower bound. Indeed, almost all known approximation algorithms operate in this manner. Under this assumption, the answer to the second question is also "no". This is established in Example 1.5, which gives an infinite family of instances on which the lower bound, of size of a maximal matching, is in fact half the size of an optimal vertex cover. In the case of linear-programming-based approximation algorithms, the analogous question will be answered by determining a fundamental quantity associated with the linear programming relaxation – its integrality gap (see Chapter 12).

The third question, of improving the approximation guarantee for vertex cover, is currently a central open problem in the field of approximation algorithms (see Section 30.1).

Example 1.5 The lower bound, of size of a maximal matching, is half the size of an optimal vertex cover for the following infinite family of instances. Consider the complete graph K_n, where n is odd. The size of any maximal matching is $(n-1)/2$, whereas the size of an optimal cover is $n-1$. □

1.2 Well-characterized problems and min–max relations

Consider decision versions of the cardinality vertex cover and maximum matching problems.

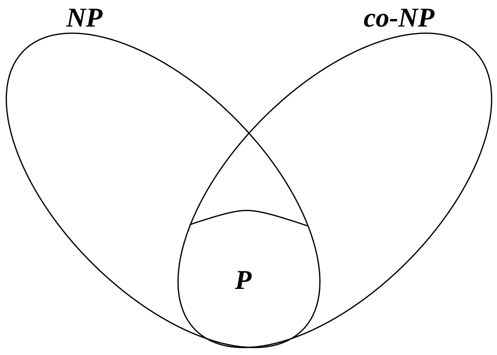
- Is the size of the minimum vertex cover in G at most k?
- Is the size of the maximum matching in G at least l?

Both these decision problems are in **NP** and therefore have Yes certificates (see Appendix A for definitions). Do these problems also have No certificates? We have already observed that the size of a maximum matching is a lower bound on the size of a minimum vertex cover. If G is bipartite, then in fact equality holds; this is the classic König-Egerváry theorem.

Theorem 1.6 *In any bipartite graph,*

$$\max_{\textit{matching } M} |M| = \min_{\textit{vertex cover } U} |U|.$$

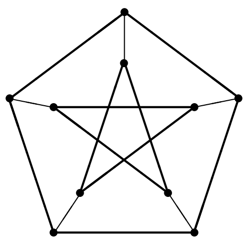
Therefore, if the answer to the first decision problem is "no", there must be a matching of size $k+1$ in G that suffices as a certificate. Similarly, a vertex cover of size $l-1$ must exist in G if the answer to the second decision problem is "no". Hence, when restricted to bipartite graphs, both vertex cover and maximum matching problems have No certificates and are in co-**NP**. In fact, both problems are also in **P** under this restriction. It is easy to see that any problem in **P** trivially has Yes as well as No certificates (the empty string suffices). This is equivalent to the statement that $\mathbf{P} \subseteq \mathbf{NP} \cap \text{co-}\mathbf{NP}$. It is widely believed that the containment is strict; the conjectured status of these classes is depicted below.



Problems that have Yes and No certificates, i.e., are in $\mathbf{NP} \cap \text{co-}\mathbf{NP}$, are said to be *well-characterized*. The importance of this notion can be gauged from the fact that the quest for a polynomial time algorithm for matching started with the observation that it is well-characterized.

Min–max relations of the kind given above provide proof that a problem is well-characterized. Such relations are some of the most powerful and beautiful results in combinatorics, and some of the most fundamental polynomial time algorithms (exact) have been designed around such relations. Most of these min–max relations are actually special cases of the LP-duality theorem (see Section 12.2). As pointed out above, LP-duality theory plays a vital role in the design of approximation algorithms as well.

What if G is not restricted to be bipartite? In this case, a maximum matching may be strictly smaller than a minimum vertex cover. For instance, if G is simply an odd length cycle on $2p+1$ vertices, then the size of a maximum matching is p, whereas the size of a minimum vertex cover is $p+1$. This may happen even for graphs having a perfect matching, for instance, the Petersen graph:



This graph has a perfect matching of cardinality 5; however, the minimum vertex cover has cardinality 6. One can show that there is no vertex cover of size 5 by observing that any vertex cover must pick at least $p+1$ vertices from an odd cycle of length $2p+1$, just to cover all the edges of the cycle, and the Petersen graph has two disjoint cycles of length 5.

Under the widely believed assumption that $\mathbf{NP} \neq \text{co-}\mathbf{NP}$, $\mathbf{NP}$-hard problems do not have No certificates. Thus, the minimum vertex cover problem in general graphs, which is $\mathbf{NP}$-hard, does not have a No certificate, assuming $\mathbf{NP} \neq \text{co-}\mathbf{NP}$. The maximum matching problem in general graphs is in $\mathbf{P}$. However, the No certificate for this problem is not a vertex cover, but a more general structure: an odd set cover.

An *odd set cover* C in a graph $G = (V, E)$ is a collection of disjoint odd cardinality subsets of V, $S_1, \ldots, S_k$, and a collection $v_1, \ldots, v_l$ of vertices such that each edge of G is either incident at one of the vertices v_i or has both endpoints in one of the sets S_i. The *weight* of this cover C is defined to be $w(C) = l + \sum_{i=1}^{k} (|S_i| - 1)/2$. The following min–max relation holds.

Theorem 1.7 *In any graph,* $$\max_{\text{matching } M} |M| = \min_{\text{odd set cover } C} w(C).$$

As shown above, in general graphs a maximum matching can be smaller than a minimum vertex cover. Can it be arbitrarily smaller? The answer is "no". A corollary of Theorem 1.3 is that in any graph, the size of a maximum matching is at least half the size of a minimum vertex cover. More precisely, Theorem 1.3 gives, as a corollary, the following approximate min–max relation. Approximation algorithms frequently yield such approximate min–max relations, which are of independent interest.

Corollary 1.8 *In any graph,*

$$\max_{\text{matching } M} |M| \;\le\; \min_{\text{vertex cover } U} |U| \;\le 2 \cdot \left(\max_{\text{matching } M} |M| \right).$$

Although the vertex cover problem does not have No certificate under the assumption $\mathbf{NP} \neq$ co-$\mathbf{NP}$, surely there ought to be a way of certifying that (G, k) is a "no" instance for small enough values of k. Algorithm 1.2 (more precisely, the lower bounding scheme behind this approximation algorithm) provides such a method. Let $\mathcal{A}(G)$ denote the size of vertex cover output by Algorithm 1.2. Then, $\mathrm{OPT}(G) \le \mathcal{A}(G) \le 2 \cdot \mathrm{OPT}(G)$. If $k < \mathcal{A}(G)/2$ then $k < \mathrm{OPT}(G)$, and therefore (G, k) must be a "no" instance. Furthermore, if $k < \mathrm{OPT}(G)/2$ then $k < \mathcal{A}(G)/2$. Hence, Algorithm 1.2 provides a No certificate for all instances (G, k) such that $k < \mathrm{OPT}(G)/2$.

A No certificate for instances (I, B) of a minimization problem Π satisfying $B < \mathrm{OPT}(I)/\alpha$ is called a *factor α approximate No certificate.* As in the case of normal Yes and No certificates, we do not require that this certificate be polynomial time computable. An α factor approximation algorithm $\mathcal{A}$ for Π provides such a certificate. Since $\mathcal{A}$ has polynomial running time, this certificate is polynomial time computable. In Chapter 27 we will show an intriguing result – that the shortest vector problem has a factor n approximate No certificate; however, a polynomial time algorithm for constructing such a certificate is not known.

1.3 Exercises

1.1 Give a factor 1/2 algorithm for the following.

Problem 1.9 (Acyclic subgraph) Given a directed graph $G = (V, E)$, pick a maximum cardinality set of edges from E so that the resulting subgraph is acyclic.
Hint: Arbitrarily number the vertices and pick the bigger of the two sets, the forward-going edges and the backward-going edges. What scheme are you using for upper bounding OPT?

1.2 Design a factor 2 approximation algorithm for the problem of finding a minimum cardinality maximal matching in an undirected graph.
Hint: Use the fact that any maximal matching is at least half the maximum matching.

1.3 (R. Bar-Yehuda) Consider the following factor 2 approximation algorithm for the cardinality vertex cover problem. Find a depth first search tree in the given graph, G, and output the set, say S, of all the nonleaf vertices of this tree. Show that S is indeed a vertex cover for G and $|S| \leq 2 \cdot \mathrm{OPT}$.
Hint: Show that G has a matching of size $|S|$.

1.4 Perhaps the first strategy one tries when designing an algorithm for an optimization problem is the greedy strategy. For the vertex cover problem, this would involve iteratively picking a maximum degree vertex and removing it, together with edges incident at it, until there are no edges left. Show that this algorithm achieves an approximation guarantee of $O(\log n)$. Give a tight example for this algorithm.
Hint: The analysis is similar to that in Theorem 2.4.

1.5 A maximal matching can be found via a greedy algorithm: pick an edge, remove its two endpoints, and iterate until there are no edges left. Does this make Algorithm 1.2 a greedy algorithm?

1.6 Give a lower bounding scheme for the arbitrary cost version of the vertex cover problem.
Hint: Not easy if you don't use LP-duality.

1.7 Let $A = \{a_1, \ldots, a_n\}$ be a finite set, and let "$\leq$" be a relation on A that is reflexive, antisymmetric, and transitive. Such a relation is called a *partial ordering of* A. Two elements $a_i, a_j \in A$ are said to be *comparable* if $a_i \leq a_j$ or $a_j \leq a_i$. Two elements that are not comparable are said to be *incomparable.* A subset $S \subseteq A$ is a *chain* if its elements are pairwise comparable. If the elements of S are pairwise incomparable, then it is an *antichain.* A *chain (antichain) cover* is a collection of chains (antichains) that are pairwise disjoint and cover A. The size of such a cover is the number of chains (antichains) in it. Prove the following min–max result. The size of a longest chain equals the size of a smallest antichain cover.
Hint: Let the size of the longest chain be m. For $a \in A$, let $\phi(a)$ denote the size of the longest chain in which a is the smallest element. Now, consider the partition of A, $A_i = \{a \in A \mid \phi(a) = i\}$, for $1 \leq i \leq m$.

1.8 (Dilworth's theorem, see [195]) Prove that in any finite partial order, the size of a largest antichain equals the size of a smallest chain cover.
Hint: Derive from the König-Egerváry Theorem. Given a partial order on n-element set A, consider the bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$ and $(u_i, v_j) \in E$ iff $a_i \leq a_j$.

The next ten exercises are based on Appendix A.

1.9 Is the following an **NP**-optimization problem? Given an undirected graph $G = (V, E)$, a cost function on vertices $c : V \to \mathbf{Q}^+$, and a positive integer k, find a minimum cost vertex cover for G containing at most k vertices.
Hint: Can valid instances be recognized in polynomial time (such an instance must have at least one feasible solution)?

1.10 Let $\mathcal{A}$ be an algorithm for a minimization **NP**-optimization problem Π such that the expected cost of the solution produced by $\mathcal{A}$ is $\leq \alpha\mathrm{OPT}$, for a constant $\alpha > 1$. What is the best approximation guarantee you can establish for Π using algorithm $\mathcal{A}$?
Hint: A guarantee of $2\alpha - 1$ follows easily. For guarantees arbitrarily close to α, run the algorithm polynomially many times and pick the best solution. Apply Chernoff's bound.

1.11 Show that if SAT has been proven **NP**-hard, and SAT has been reduced, via a polynomial time reduction, to the decision version of vertex cover, then the latter is also **NP**-hard.
Hint: Show that the composition of two polynomial time reductions is also a polynomial time reduction.

1.12 Show that if the vertex cover problem is in co-**NP**, then **NP** = co-**NP**.

1.13 (Pratt [222]) Let L be the language consisting of all prime numbers. Show that $L \in \mathbf{NP}$.
Hint: Consider the multiplicative group mod$n$, $Z_n^* = \{a \in \mathbf{Z}^+ \mid 1 \leq a < n \text{ and } (a, n) = 1\}$. Clearly, $|Z_n^*| \leq n - 1$. Use the fact that $|Z_n^*| = n - 1$ iff n is prime, and that Z_n^* is cyclic if n is prime. The Yes certificate consists of a primitive root of Z_n^*, the prime factorization of $n - 1$, and, recursively, similar information about each prime factor of $n - 1$.

1.14 Give proofs of self-reducibility for the optimization problems discussed later in this book, in particular, maximum matching, MAX-SAT (Problem 16.1), clique (Problem 29.15), shortest superstring (Problem 2.9), and Minimum makespan scheduling (Problem 10.1).
Hint: For clique, consider two possibilities, that v is or isn't in the optimal clique. Correspondingly, either restrict G to v and its neighbors, or remove v from G. For shortest superstring, remove two strings and replace them by a legal overlap (may even be a simple concatenation). If the length of the optimal superstring remains unchanged, work with this smaller instance. Generalize the scheduling problem a bit – assume that you are also given the number of time units already scheduled on each machine as part of the instance.

1.15 Give a suitable definition of self-reducibility for problems in **NP**, i.e., decision problems and not optimization problems, which enables you to obtain a polynomial time algorithm for finding a feasible solution given an oracle for the decision version, and moreover, yields a self-reducibility tree for instances.
Hint: Impose an arbitrary order among the atoms of a solution, e.g., for SAT, this was achieved by arbitrarily ordering the n variables.

1.16 Let Π_1 and Π_2 be two minimization problems such that there is an approximation factor preserving reduction from Π_1 to Π_2. Show that if there is an α factor approximation algorithm for Π_2 then there is also an α factor approximation algorithm for Π_1.
Hint: First prove that if the reduction transforms instance I_1 of Π_1 to instance I_2 of Π_2 then $\mathrm{OPT}_{\Pi_1}(I_1) = \mathrm{OPT}_{\Pi_2}(I_2)$.

1.17 Show that

$$L \in \mathbf{ZPP} \text{ iff } L \in (\mathbf{RP} \cap \text{co-}\mathbf{RP}).$$

1.18 Show that if $\mathbf{NP} \subseteq \text{co-}\mathbf{RP}$ then $\mathbf{NP} \subseteq \mathbf{ZPP}$.
Hint: If SAT instance ϕ is satisfiable, a satisfying truth assignment for ϕ can be found, with high probability, using self-reducibility and the co-**RP** machine for SAT. If ϕ is not satisfiable, a "no" answer from the co-**RP** machine for SAT confirms this; the machine will output such an answer with high probability.

1.4 Notes

The notion of well-characterized problems was given by Edmonds [69] and was precisely formulated by Cook [51]. In the same paper, Cook initiated the theory of **NP**-completeness. Independently, this discovery was also made by Levin [186]. It gained its true importance with the work of Karp [164], showing **NP**-completeness of a diverse collection of fundamental computational problems.

Interestingly enough, approximation algorithms were designed even before the discovery of the theory of **NP**-completeness, by Vizing [255] for the minimum edge coloring problem, by Graham [113] for the minimum makespan problem (Problem 10.1), and by Erdös [73] for the MAX-CUT problem (Problem 2.14). However, the real significance of designing such algorithms emerged only after belief in the $\mathbf{P} \neq \mathbf{NP}$ conjecture grew. The notion of an approximation algorithm was formally introduced by Garey, Graham, and Ullman [91] and Johnson [150]. The first use of linear programming in approximation algorithms was due to Lovász [192], for analyzing the greedy set cover algorithm (see Chapter 13). An early work exploring the use of randomization in the design of algorithms was due to Rabin [224] – this notion is useful in the design of approximation algorithms as well. Theorem 1.7 is due to Edmonds [69] and Algorithm 1.2 is due to Gavril [93].

For basic books on algorithms, see Cormen, Leiserson, Rivest, and Stein [54], Papadimitriou and Steiglitz [217], and Tarjan [246]. For a good treatment of min–max relations, see Lovász and Plummer [195]. For books on approximation algorithms, see Hochbaum [126] and Ausiello, Crescenzi, Gambosi, Kann, Marchetti, and Protasi [17]. Books on linear programming, complexity theory, and randomized algorithms are listed in Sections 12.5, A.6, and B.4, respectively.