国外数学名著系列

（影印版）8

Ingo Wegener

Complexity Theory

复杂性理论

科学出版社
www.sciencep.com

图字:01-2005-6731

图书在版编目(CIP)数据

复杂性理论/(德)韦格纳(Ingo,W.)著.—影印版.—北京:科学出版社,2006

(国外数学名著系列)

ISBN 7-03-016692-2

Ⅰ.复… Ⅱ.韦… Ⅲ.复杂性理论-英文 Ⅳ.0224

中国版本图书馆CIP数据核字(2005)第154385号

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com
中国科学院印刷厂 印刷
科学出版社发行 各地新华书店经销
*
2006年1月第 一 版 开本:B5(720×1000)
2006年1月第一次印刷 印张:20 1/2
印数:1—2 000 字数:380 000

定价:66.00元

(如有印装质量问题,我社负责调换(科印))

《国外数学名著系列》(影印版)序

要使我国的数学事业更好地发展起来，需要数学家淡泊名利并付出更艰苦地努力。另一方面，我们也要从客观上为数学家创造更有利的发展数学事业的外部环境，这主要是加强对数学事业的支持与投资力度，使数学家有较好的工作与生活条件，其中也包括改善与加强数学的出版工作。

从出版方面来讲，除了较好较快地出版我们自己的成果外，引进国外的先进出版物无疑也是十分重要与必不可少的。从数学来说，施普林格（Springer）出版社至今仍然是世界上最具权威的出版社。科学出版社影印一批他们出版的好的新书，使我国广大数学家能以较低的价格购买，特别是在边远地区工作的数学家能普遍见到这些书，无疑是对推动我国数学的科研与教学十分有益的事。

这次科学出版社购买了版权，一次影印了 23 本施普林格出版社出版的数学书，就是一件好事，也是值得继续做下去的事情。大体上分一下，这 23 本书中，包括基础数学书 5 本，应用数学书 6 本与计算数学书 12 本，其中有些书也具有交叉性质。这些书都是很新的，2000 年以后出版的占绝大部分，共计 16 本，其余的也是 1990 年以后出版的。这些书可以使读者较快地了解数学某方面的前沿，例如基础数学中的数论、代数与拓扑三本，都是由该领域大数学家编著的“数学百科全书”的分册。对从事这方面研究的数学家了解该领域的前沿与全貌很有帮助。按照学科的特点，基础数学类的书以“经典”为主，应用和计算数学类的书以“前沿”为主。这些书的作者多数是国际知名的大数学家，例如《拓扑学》一书的作者诺维科夫是俄罗斯科学院的院士，曾获“菲尔兹奖”和“沃尔夫数学奖”。这些大数学家的著作无疑将会对我国的科研人员起到非常好的指导作用。

当然，23 本书只能涵盖数学的一部分，所以，这项工作还应该继续做下去。更进一步，有些读者面较广的好书还应该翻译成中文出版，使之有更大的读者群。

总之，我对科学出版社影印施普林格出版社的部分数学著作这一举措表示热烈的支持，并盼望这一工作取得更大的成绩。

王　元

2005 年 12 月 3 日

Preface to the Original German Edition

At least since the development of the theory of NP-completeness, complexity theory has become a central area of instruction and research within computer science. The NP $\neq$ P-problem represents one of the great intellectual challenges of the present. In contrast to other areas within computer science, where it is often suggested that nearly all problems are solvable with the aid of computers, the goals of complexity theory include showing what computers cannot do. Delineating the boundary between problems that can be efficiently solved and those that can only be solved with an unreasonable amount of resources is a practically relevant question, but so is the structural question of what determines the complexity or "complicatedness" of problems.

The development of complexity theory is presented in this book as essentially a reaction to algorithmic development. For this reason, the investigation of practically important optimization problems plays a predominant role. From this algorithmic perspective, reduction concepts can be thought of as methods to solve problems with the help of algorithms for other problems. From this it follows conversely that we can derive the difficulty of a problem from the difficulty of other problems.

In this book we choose an unusual approach to the central concept of nondeterminism. The usual description, based on computers that guess a correct computation path or for which a suitable computation path exists, is often confusing to students encountering nondeterminism for the first time. Here this description is replaced with an introduction to randomized algorithms. Nondeterminism is then simply the special case of one-sided error with an error-rate that may be larger than is tolerable in applications. In this presentation, nondeterministic algorithms can be run on normal computers, but do not provide a satisfactory solution to problems. Based on experience, we are hopeful that this algorithmic approach will make it simpler for students to grasp the concept of nondeterminism.

Since this is not intended to be a research monograph, the content has been limited to results that are important and useful for students of computer science. In particular, this text is aimed at students who want an introduction

to complexity theory but do not necessarily plan to specialize in this area. For this reason, an emphasis has been placed on informal descriptions of the proof ideas, which are, of course, followed by complete proofs. The emphasis is on modern themes like the PCP-theorem, approximation problems, randomization, and communication complexity at the expense of structural and abstract complexity theory.

The first nine chapters describe the foundation of complexity theory. Beyond that, instructors can choose various emphases:

- Chapters 10, 13, and 14 describe a more classically oriented introduction to complexity theory,
- Chapters 11 and 12 treat the complexity of approximation problems, and
- Chapters 14, 15, and 16 treat the complexity of Boolean functions.

Many ideas have come together in this text that arose in conversations. Since it is often no longer possible to recall where, when, and with whom these conversations were held, I would like to thank all those who have discussed with me science in general and complexity theory in particular. Many thanks to Beate Bollig, Stefan Droste, Oliver Giel, Thomas Hofmeister, Martin Sauerhoff, and Carsten Witt, who read the [original German] manuscript and contributed to improvements through their critical comments, and to Alice Czerniejewski, Danny Rozynski, Marion Scheel, Nicole Skaradzinski, and Dirk Sudholt for their careful typesetting.

Finally, I want to thank Christa for not setting any limits on the time I could spend on this book.

Dortmund/Bielefeld, January 2003 *Ingo Wegener*

Preface to the English Edition

This book is the second translation project I have undertaken for Springer. My goal each time has been to produce a text that will serve its new audience as well as the original book served its audience. Thus I have tried to mimic as far as possible the style and "flavor" of the original text while making the necessary adaptations. At the same time, a translation affords an opportunity to make some improvements, which I have done in consultation with the original author. And so, in some sense, the result is a translation of a second edition that was never written.

Most of the revisions to the book are quite minor. Some bibliography items have been added or updated; a number of German sources have been deleted. Occasionally I have added or rearranged a paragraph, or included some additional detail, but for the most part I have followed the original quite closely. Where I found errors in the original, I have tried to fix them; I hope I have corrected more than I have introduced.

It is always a good feeling to come to the end of a large project like this one, and in looking back on the project there are always a number of people to thank. Much of the work done to prepare the English edition of this book was done while visiting the University of Ulm in the first half of 2004. The last revisions and final touches were completed during my subsequent visit at the University of Michigan. I would like to thank all my colleagues at both institutions for their hospitality during these visits.

A writer is always the worst editor of his own writing, so for reading portions of the text, identifying errors, and providing various suggestions for improvement, I want to thank Beate Bollig, Stefan Droste, Jeremy Frens, Judy Goldsmith, André Gronemeier, Jens Jägersküpper, Thomas Jansen, Marcus Schaefer, Tobias Storch, and Dieter van Melkebeek, each of whom read one or more chapters. In addition, my wife, Pennylyn, read nearly the entire manuscript. Their volunteered efforts have helped to ensure a more accurate and stylistically consistent text. A list of those (I hope few) errors that have escaped detection until after the printing of the book will be available at

ls2-www.cs.uni-dortmund.de/monographs/ct

Finally, a special thanks goes to Ingo Wegener, who not only wrote the original text but also responded to my comments and questions, and read the English translation with a careful eye for details; and to Hermann Engesser and Dorothea Glaunsinger at Springer for their encouragement, assistance, and patience, and for a fine *Kaffeestunde* on a sunny afternoon in Heidelberg.

Ann Arbor, January 2005 *Randall Pruim*

It is possible to write a research monograph in a non-native language. In fact, I have done this before. But a textbook with a pictorial language needs a native speaker as translator. Moreover, the translator should have a good feeling for the formulations and a background to understand and even to shape and direct the text. Such a person is hard to find, and it is Randall Pruim who made this project possible and, as I am convinced, in a perfect way. Indeed, he did more than a translation. He found some mistakes and corrected them, and he improved many arguments. Also thanks to Dorothea Glaunsinger and Hermann Engesser from Springer for their enthusiastic encouragement and for their suggestion to engage Randall Pruim as translator.

Bielefeld/Dortmund, January 2005 *Ingo Wegener*

Contents

1

Introduction

1.1 What Is Complexity Theory?

Complexity theory – is it a discipline for theoreticians who have no concern for "the real world" or a central topic of modern computer science?

In this introductory text, complexity theory is presented as an active area of computer science with results that have implications for the development and use of algorithms. Our study will lead to insights into the structure of important optimization problems and will explore the borders of what is algorithmically "possible" with reasonable resources. Since this text is also especially directed toward those who do not wish to make complexity theory their specialty, results that do not (yet) have a connection to algorithmic applications will be omitted.

The areas of complexity theory on the one hand and of the design and analysis of efficient algorithms on the other look at algorithmic problems from two opposing perspectives. An efficient algorithm can be directly applied to solve a problem and is itself a proof of the efficient solvability of the problem. In contrast, in complexity theory the goal is to prove that difficult problems cannot be solved with modest resources. Bearers of bad news are seldom welcome, and so it is that the results of complexity theory are more difficult to communicate than a better algorithm for an important problem. Those who do complexity theory are often asked such questions as

- "Why are you pleased with a proof that a problem is algorithmically difficult? It would be better if it had an efficient algorithmic solution."
- "What good are these results? For my particular applied problem I need an algorithmic solution. Now what do I do?"

Naturally, it would be preferable if a problem proved to be efficiently algorithmically solvable. But whether or not this is the case is not up to us. Once we have agreed upon the rules of the game (roughly: computers, but more about that later), every problem has a well-defined algorithmic complexity. Complexity theory and algorithm theory are both striving to estimate this

algorithmic complexity and so to "discover the truth". In this sense, the joy over a proof that a problem is not efficiently solvable is, just like the joy over the design of an efficient algorithm, the joy of finding out more about the true algorithmic complexity.

Of course, our reaction to the discovery of truths does depend on whether hopes were fulfilled or fears confirmed. What are the consequences when we find out that the problem we are investigating is not efficiently solvable? First, there is the obvious and very practical consequence that we can with good reason abandon the search for an efficient algorithm. We need no longer waste our time with attempts to obtain an unreachable goal. We are familiar with this from other sciences as well. Reasonable people no longer build "perpetual motion machines", and they no longer try to construct from a circle, using only straight edge and compass, a square with the same area (the proverbial quadrature of the circle). In general, however, people have a hard time with impossibility results. This can be seen in the large number of suggested designs for perpetual motion machines and the large number of attempts to square a circle that are still being made.

Once we have understood that we must accept negative results as well as positive results, and that they save us unnecessary work, we are left with the question of what to do. In the end, we are dealing with an algorithmic problem the solution to which is important for some particular application. Fortunately, problems in most applications are not unalterably determined. It is often tempting to formulate a problem in a very general form and to place very strict demands on the quality of the solution. If such a general formulation has an efficient solution, great. But when this is not the case, we can often specialize the problem (graphs that model street systems will have low degree because there is a limit on the number of streets that can meet at a single intersection), or perhaps a weaker form of solution will suffice (almost optimal may be good enough). In this way we come up with new problems which are perhaps efficiently algorithmically solvable. And so impossibility proofs (negative results) help us find the problems that are (perhaps "just barely") efficiently solvable.

So complexity theory and the design and analysis of efficient algorithms are the two areas of computer science which together fathom the borders between what can and cannot be done algorithmically with realistic resource requirements. There is, of course, a good deal of "cross-pollination" between the two areas. Often attempts to prove the impossibility of an efficient solution to a problem have so illuminated the structure of the problem that efficient algorithms have been the result. On the other hand, failed attempts to design an efficient algorithm often reveal just where the difficulty of a particular problem lies. This can lead to ideas for proving the difficulty of the problem. It is very often the case that one begins with a false conjecture about the degree of difficulty of a problem, so we can expect to encounter startling results in our study of the complexity of problems.

As a result of this introductory discussion we maintain that

The goal of complexity theory is to prove for important problems that their solutions require certain minimum resources. The results of complexity theory have specific implications for the development of algorithms for practical applications.

We have up until now been emphasizing the relationship between the areas of complexity theory and algorithm design. Now, however, we want to take a look at the differences between these areas. When designing an algorithm we "only" need to develop and analyze *one* algorithm. This provides an *upper bound* for the minimal resource requirements with which the problem can be solved. Complexity theory must provide *lower bounds* for the minimally necessary resource requirements that *every* algorithm that solves the problem must use. For the proof of an upper bound, it is sufficient to design and analyze a *single* algorithm (and algorithms are often designed to support the subsequent analysis). Every lower bound, on the other hand, is a statement about *all* algorithms that solve a particular problem. The set of all algorithms for a problem is not a very structured set. Its only structural characteristic is that the problem be solved. How can we make use of this characteristic? An obvious way to start is to derive from the structure of the problem statements that restrict the set of algorithms we must consider. A specific example: It seems clear that the best algorithms for matrix multiplication do not begin by subtracting matrix elements from each other. But how does one prove this? Or is a proof unnecessary, since the claim is so obvious? Quite the opposite: The best algorithms known for matrix multiplication do in fact begin by subtracting matrix elements (see, for example, Schönhage, Grotefeld, and Vetter (1999)). This clearly shows the danger in drawing very "obvious" but false conclusions. Therefore,

In order to prove that the solution of a particular problem requires certain minimal resources, all algorithms for the problem must be considered. This is the source of the main difficulty that impedes achieving the goals of complexity theory.

We now know what kind of results we desire, and we have indicated that they are difficult to come by. It sounds as if we want to excuse in advance the absence of results. This is indeed the case:

None of the most important problems in complexity theory have been solved, but along the way to answering the central questions many notable results have been achieved.

How do we imagine this situation? The cover of the classic book by Hopcroft and Ullman (1979), which includes an introduction to complexity theory, shows a picture in which a curtain in front of the collection of truths

of complexity theory is being lifted with the help of various results, thus allowing a clear view of the results. From our perspective of complexity theory, the curtain has so far only been pushed aside a bit at the edges, so that we can clearly see some "smaller truths". Otherwise, the opaque curtain has been replaced by a thinner curtain through which we can recognize a large portion of the truth, but only in outline and with no certainty that we are not falling prey to an optical illusion.

What does that mean concretely? Problems that are viewed as difficult have not actually been proved to be difficult, but it has been shown that thousands of problems are essentially equally difficult (in a sense that will be made precise later). An efficient solution to any one of these thousands of problems implies an efficient solution to all the others. Or stated another way: a proof that any one of these problems is not efficiently solvable implies that none of them is. Thousands of secrets have joined together to form one great mystery, the unmasking of which reveals all the secrets. In this sense, each of these secrets is just as central as every other and just as important as the great mystery, which we will later refer to as the NP $\neq$ P-problem. In contrast to many other areas of computer science,

Complexity theory has in the NP $\neq$ P*-problem a central challenge.*

The advantage of such an important and central problem is that along the way to its solution many important results, methods, and even new research areas are discovered. The disadvantage is that the solution of the central problem may be a long time in coming. We can learn something of this from the 350-year search for a proof of Fermat's Last Theorem (Singh (1998) is recommended for more about that topic). Along the way to the solution, deep mathematical theories were developed but also many false paths were followed. Only because of the notoriety of Fermat's Last Theorem was so much effort expended toward the solution to the problem. The NP $\neq$ P-problem has taken on a similar role in computer science – but with an unfortunate difference: Fermat's Last Theorem (which says that there are no natural numbers x, y, z, and n with $n \geq 3$ such that $x^n + y^n = z^n$) can be understood by most people. It is fascinating that a conjecture that is so simple to formulate occupied the world of mathematics for centuries. For the role of computer science, it would be nice if it were equally simple to explain to a majority of people the complexity class P and especially NP, and the meaning of the NP $\neq$ P-problem. Alas, this is not the case.

We will see that in the vicinity of the NP $\neq$ P-problem important and beautiful results have been achieved. But we must also fear that much time may pass before the NP $\neq$ P-problem is solved. For this reason, it is not necessarily the best strategy to aim directly for a solution to the problem. Yao (2001) compared our starting position to the situation of those who 200 years ago dreamed of reaching the moon. The strategy of climbing the nearest tree or mountain brings us closer to the moon, but it doesn't really bring us any closer to the goal of reaching the moon. The better strategy was to develop

ever better means of transportation (bicycles, automobiles, airplanes, rockets). Each of these intermediate steps represented an earth moving discovery. So it is with complexity theory at the beginning of the third millennium: we must search for intermediate steps and follow suitable paths, even though we can never be certain that they will lead to our goal.

Just as those who worked on Fermat's Last Theorem were "sure" that the conjecture was true, so it is that today the experts believe that $\mathsf{NP} \neq \mathsf{P}$ and, therefore, that all of the essentially equally difficult problems mentioned above are not efficiently solvable. Why is this so? From the opposite assumption that $\mathsf{NP} = \mathsf{P}$ one can derive consequences that contradict all our convictions, even though they have not been proven false. Strassen (1996) has gone so far as to elevate the $\mathsf{NP} \neq \mathsf{P}$-conjecture above the status of a mathematical conjecture and compared it with a physical law (such as $E = mc^2$). This, by the way, opens up the possibility that the hypothesis that $\mathsf{NP} \neq \mathsf{P}$ is true but not provable with our proof techniques. But at this point we are far from being able to discuss this background seriously. Our main conclusion is that it is reasonable to build a theory under the hypothesis that $\mathsf{NP} \neq \mathsf{P}$.

Many results in complexity theory assume solidly based but unproven hypotheses, such as $\mathsf{NP} \neq \mathsf{P}$.

But what if $\mathsf{NP} = \mathsf{P}$? Well, then we must make fundamental modifications to many of our intuitions. Many of the results discussed here would in this case have other interpretations, but most would not become worthless. In general, complexity theory forms an intellectual challenge that differs from the demands of other areas of computer science. Complexity theory takes its place in the scientific landscape among those disciplines that

seek to probe the boundaries of what is possible with available resources.

Here the resources are such things as computation time and storage space. Anyone who is interested in the boundaries of what is (and is not) practically feasible with computers will find that complexity theory provides important answers. But those who come to complexity theory only wanting to know pragmatically if the problem they are interested in can be efficiently solved have also come to the right place.

1.2 Didactic Background

The main goal of this text is to provide as many as possible with a comfortable introduction to modern complexity theory. To this end a number of decisions were made with the result that this text differs from other books on the subject.

Since complexity theory is a polished theory with many branches, some selection of topics is unavoidable. In our selection, we have placed a premium

on choosing topics that have a concrete relationship to algorithmic problems. After all, we want the importance of complexity theory for modern computer science to be clear. This comes at the cost of structural and abstract branches of complexity theory, which are largely omitted. In Section 1.3 we discuss in more detail just which topics are covered.

We have already discussed the difficulties of dealing with negative results and the relationship to the area of algorithm design. With a consistent perspective that is markedly algorithmic, we will – whenever it is possible and reasonable – present first positive results and only then derive consequences of negative results. For this reason, we will often quantify results which are typically presented only qualitatively.

In the end, it is the concept of nondeterminism that presents a large hurdle that one must clear in order to begin the study of complexity theory. The usual approach is to first describe nondeterministic computers which "guess" the correct computation path, and therefore can not actually be constructed. We have chosen instead to present randomization as the key concept. Randomized algorithms can be realized on normal computers and the modern development of algorithms has clearly shown the advantages of randomized algorithms (see Motwani and Raghavan (1995)). Nondeterminism then becomes a special case of randomization and therefore an algorithmically realizable concept, albeit one with an unacceptable probability of error (see Wegener (2002)). Using this approach it is easy to derive the usual characterizations of nondeterminism later.

We will, of course, give complete and formal proofs of our results, but often there are ugly details that make the proofs long and opaque. The essential ideas, however, are usually shorter to describe and much clearer. So we will include, in addition to the proofs, discussions of the ideas, methods, and concepts involved, in the hope that the interplay of all components will ease the introduction to complexity theory.

1.3 Overview

In Section 1.1 we simplified things by assuming that a problem is either algorithmically difficult or efficiently solvable. All concepts that are not formally defined must be uniquely specified. This begins already with the concept of an algorithmic problem. Doesn't the difficulty of a problem depend on just how one formulates the problem and on the manner in which the necessary data are made available? In Chapter 2 we will clarify essential notions such as algorithmic problem, computer, computation time, and algorithmic complexity. So that we can talk about some example problems, several important algorithmic problems and their variants will also be introduced and motivated. To avoid breaking up the flow of the text, a thorough introduction to O-notation has been relegated to the appendix.