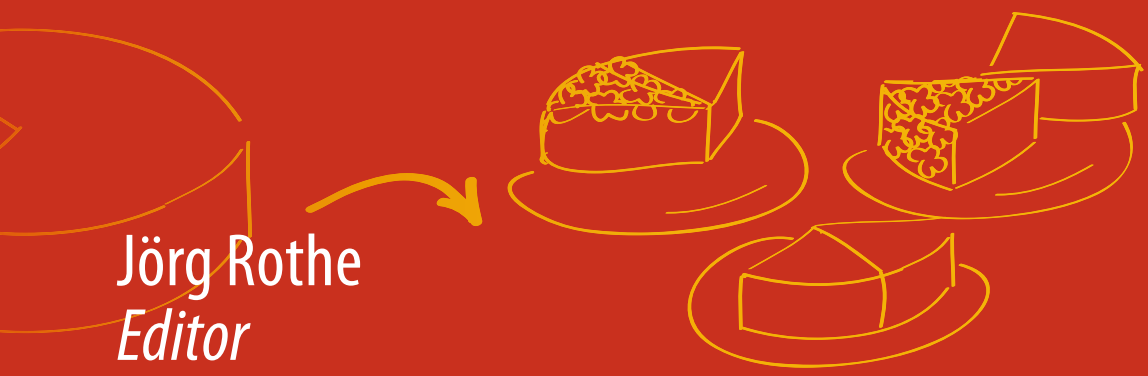
Springer Texts in Business and Economics



Jörg Rothe
Editor

Economics and Computation

An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division

Springer

Springer Texts in Business and Economics

More information about this series at http://www.springer.com/series/10099

Jörg Rothe
Editor

Economics and Computation

An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division

Illustrations by Irene Rothe

Springer

Editor
Jörg Rothe
Department of Computer Science
Heinrich-Heine-Universität Düsseldorf
Düsseldorf, Germany

ISSN 2192-4333 ISSN 2192-4341 (electronic)
Springer Texts in Business and Economics
ISBN 978-3-662-47903-2 ISBN 978-3-662-47904-9 (eBook)
DOI 10.1007/978-3-662-47904-9

Library of Congress Control Number: 2015948869

Springer Heidelberg New York Dordrecht London

Illustrations by Irene Rothe

Printed on acid-free paper

Springer-Verlag GmbH Berlin Heidelberg is part of Springer Science+Business Media (www.springer.com)

Foreword by Matthew O. Jackson and Yoav Shoham

One of the most exciting, interesting, and important areas of interdisciplinary research over the past two decades has been at the juncture of computer science and economics, breathing new life into game-theoretic analyses of the many mechanisms and institutions that pervade our lives. It is an area that gives rise to fascinating intellectual problems, that are at the same time highly relevant to electronic commerce and other significant areas of life in the 21st century. In particular, the focus on complexity has not only forced important practical considerations to be taken into account in designing systems from elections to auctions, but has also provided new insights into why we see specific institutional features rather than more complex cousins that might be better from an unconstrained theoretical perspective.

The literature has rapidly advanced on this subject, and it is getting to the point where it is increasingly difficult to keep track of what is known and what is not, and what general insights are emerging. This volume fills a critical void. While there exist other, excellent publications covering some parts of this growing and sprawling literature, notably missing has been coverage of the areas of computational social choice and fair division, the focus of this volume. Its coverage is broad and encompassing: from voting systems, to judgment aggregation, to the allocation of indivisible goods, to the age-old problem of fair division viewed through a new lens. Moreover, it provides a very accessible introduction that should be required reading for anyone venturing into the area for the first time. We offer our congratulations to the editor and the authors for this impressive achievement.

Matthew O. Jackson and Yoav Shoham
Stanford University, Palo Alto, USA
May 2015

Preface by the Editor

Our work on this book has started in 2012, shortly after its German predecessor, *"Einführung in Computational Social Choice: Individuelle Strategien und kollektive Entscheidungen beim Spielen, Wählen und Teilen"* [510], was published by Spektrum Akademischer Verlag. However, the present book is not merely a translation of this former book into English: Each of its chapters is considerably more comprehensive than in the predecessor, there is one additional chapter (namely, Chapter 5 on the complexity of manipulative actions in single-peaked societies), and instead of having only four authors, this book has been written by ten authors. That is why this book assigns authors specifically to the chapters they have written. While each of the four authors of the German book has been working hard to extend and improve her or his chapter(s), I am proud and grateful to have found and persuaded six additional authors, each an internationally renowned expert of her or his field, to contribute to this book.

Here is some information on each of the ten authors and on their chapters:

- Dorothea Baumeister[1] from Heinrich-Heine-Universität Düsseldorf, Germany, has coauthored Chapter 4 on preference aggregation by voting and Chapter 6 on judgment aggregation,
- Edith Elkind from University of Oxford, UK, has coauthored Chapter 3 on cooperative game theory,
- Gábor Erdélyi[2] from University of Siegen, Germany, has coauthored Chapter 6 on judgment aggregation,
- Piotr Faliszewski[3] from AGH University of Science and Technology in Kraków, Poland, has coauthored Chapter 2 on noncooperative game theory,
- Edith Hemaspaandra[4] from Rochester Institute of Technology, USA, has coauthored Chapter 5 on the complexity of manipulative actions in single-peaked societies,

[1] Her work has been supported in part by an NRW grant for gender-sensitive universities supporting her as a junior professor for Computational Social Choice and by the project "Online Partizipation," both funded by the NRW Ministry for Innovation, Science, and Research, and by DFG grant RO-1202/15-1.

[2] His work has been supported in part by DFG grant ER-738/2-1, by "Förderverein des FB Wirtschaftswissenschaften, Wirtschaftsinformatik und Wirtschaftsrecht der Universität Siegen e.V.," and by the Short-term Scientific Mission program of COST Action IC1205 on Computational Social Choice.

[3] His work has been supported in part by AGH University grant 11.11.230.124.

[4] Her work has been supported in part by NSF grant CCF-1101452 and by COST Action IC1205 on Computational Social Choice.

- Lane A. Hemaspaandra[5] from University of Rochester, USA, also has coauthored Chapter 5 on the complexity of manipulative actions in single-peaked societies,
- Jérôme Lang[6] from CNRS-LAMSADE, Université Paris-Dauphine, France, has coauthored Chapter 8 on fair division of indivisible goods,
- Claudia Lindner from University of Manchester, UK, has coauthored Chapter 7 on cake-cutting: fair division of divisible goods,
- Irene Rothe from Bonn-Rhein-Sieg University of Applied Sciences, Germany, has coauthored Chapter 2 on noncooperative game theory, and
- Jörg Rothe[7] from Heinrich-Heine-Universität Düsseldorf, Germany, has written introductory Chapter 1 and has coauthored Chapters 2–8.

The subject of this book, generally speaking, is collective decision making in three areas, each having both an economical and a computational dimension. Accordingly, the book is divided into three parts:

Part I (Playing Successfully) is concerned with algorithmic game theory, where Chapter 2 introduces to noncooperative games and Chapter 3 to cooperative games, focusing on their computational aspects.

Part II (Voting and Judging) introduces to computational social choice. Chapter 4 is concerned with preference aggregation by voting, first providing some background from social choice theory and then focusing on the complexity of determining (possible and necessary) winners in elections and of manipulative actions to influence their outcomes. Chapter 5 sheds some light on the complexity of manipulative actions in single-peaked societies. Chapter 6 introduces to the emerging field of judgment aggregation, again with a focus on the complexity of related problems.

Part III (Fair Division) deals with mechanisms of fair division among a number of players, both for fairly dividing a divisible good (an area known as "cake-cutting") in Chapter 7 and for fairly dividing indivisible goods in Chapter 8, again focusing on computational aspects.

These three parts are preceded by a brief introduction to playing, voting, and dividing in Chapter 1, which also gives the needed notions from computational complexity theory to be used throughout the book.

[5] His work has been supported in part by NSF grants CCF-0915792 and CCF-1101479 and by COST Action IC1205 on Computational Social Choice.

[6] His work has been supported in part by ANR Project CoCoRICo-CoDec, by the DAAD-PPP/PHC PROCOPE program entitled "Fair Division of Indivisible Goods: Incomplete Preferences, Communication Protocols and Computational Resistance to Strategic Behavior," and by COST Action IC1205 on Computational Social Choice.

[7] His work has been supported in part by DFG grants RO-1202/14-1 and RO-1202/15-1, the DAAD-PPP/PHC PROCOPE program entitled "Fair Division of Indivisible Goods: Incomplete Preferences, Communication Protocols and Computational Resistance to Strategic Behavior," by the project "Online Partizipation" funded by the NRW Ministry for Innovation, Science, and Research, and by COST Action IC1205 on Computational Social Choice.

This book provides an accessible introduction to the areas mentioned above, which makes it a valuable source for teaching. Indeed, I have taught courses related to most of the single chapters of this book at my university since 2009, and so have the other authors at their universities. A noteworthy feature of this book is that its most important concepts and ideas are introduced not only in formal, technical terms but are also accompanied by numerous examples (usually told as a story from everyday life and featuring the same main characters throughout the book—have a look at the many gray boxes for the stories and at Figure 1.1 on page 2 for the book's main characters!), a total of 119 figures and 59 tables, and wonderful illustrations created by Irene Rothe.

Care has been taken to unify notation and formalism throughout the book, and there are plenty of cross-references between the chapters to point the reader to identical or closely related notions in different contexts. Moreover, an extensive bibliography with 625 references and a comprehensive index of more than 25 pages will be helpful for the reader. Note that authors are indexed even if their names are hidden in "et al." or in a plain reference without author names.

Regarding personal pronouns, referring to individual players, voters, candidates, judges, or agents by "she" alone or "he" alone would be inappropriate, and referring to them as "it" is simply wrong and ugly; therefore, we follow the approach of Chalkiadakis, Elkind, and Wooldridge [145] who promote an interleaved, (semi-)random usage of "she" and "he."

On a personal note, I'm deeply indebted to many individuals for their help in proofreading the single chapters of this book. They have done a great job, and my collective thanks go to Dorothea Baumeister, Piotr Faliszewski, Daniel Neugebauer, Nhan-Tam Nguyen, Anja Rey, and Lena Schend.

I have been working on parts of this book during a number of research visits to Université Paris-Dauphine, Stanford University (where I spent my sabbatical in 2013), Rochester Institute of Technology, and University of Rochester, and I'm grateful to the hosts of these visits, Jérôme Lang, Yoav Shoham, Edith Hemaspaandra, and Lane A. Hemaspaandra, for their warm hospitality. Last but not least, I thank Matthew O. Jackson and Yoav Shoham from Stanford University for reading an early draft of this book and for contributing to its preface.

Jörg Rothe
Düsseldorf, Germany
May 2015

Contents

Contributors

Dorothea Baumeister, Heinrich-Heine-Universität Düsseldorf, Germany

Edith Elkind, University of Oxford, UK

Gábor Erdélyi, University of Siegen, Germany

Piotr Faliszewski, AGH University of Science and Technology, Kraków, Poland

Edith Hemaspaandra, Rochester Institute of Technology, USA

Lane A. Hemaspaandra, University of Rochester, USA

Jérôme Lang, CNRS-LAMSADE, Université Paris-Dauphine, France

Claudia Lindner, University of Manchester, UK

Irene Rothe, Bonn-Rhein-Sieg University of Applied Sciences, Germany

Jörg Rothe, Heinrich-Heine-Universität Düsseldorf, Germany

Chapter 1
Playing, Voting, and Dividing

Jörg Rothe

Playing, voting, and dividing are three everyday activities we all are familiar with. Having their personal chances of winning, their individual preferences, and their private valuations in mind, the players, voters, and dividers follow their individual strategies each. While everyone first and foremost is selfishly interested in his own advantage only, from the interplay of all actors' individual interests, strategies, and actions there will emerge a *collective* decision, an outcome of the game with winnings or losings for all players, an elected president ruling over all voters, or a division of the goods among all parties concerned. By the end of the day, there will be winners and losers.

However, it is just one thing to maximize one's individual profit in a game or in a division of goods, or to make one's favorite candidate win. It is quite another thing to look at this from a more global perspective: Is it possible to find mechanisms that increase the *social* or *societal* welfare and thus are beneficial to all and not only to single individuals? So as to help, for example, a whole family—both the parents and their children—choosing a consensual weekend trip destination by voting. Or, so as to help three siblings playing a parlor game to choose their strategies so optimally that none of them could do better by picking another strategy without making any other player be worse off at the same time. Or, so as to help them dividing a cake afterwards, whose single pieces are valued differently by everyone, in a way that no one will envy anyone else for their portion. Strategy-proofness of such mechanisms and procedures is another important goal. If someone tries to get an unfair advantage by choosing insincere strategies, is it possible to prevent that from happening by using a strategy-proof mechanism?

This book introduces to three emerging, interdisciplinary fields at the interface of economics and the political and social sciences on the one hand and (various fields of) computer science and mathematics on the other hand:

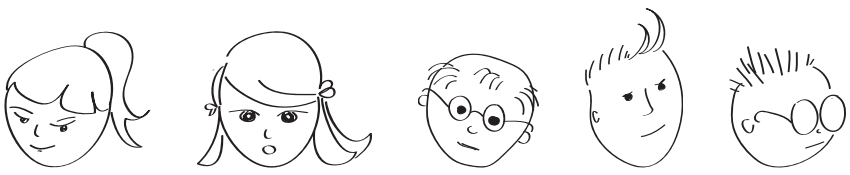
- *Algorithmic game theory* will be handled in Chapters 2 and 3. Starting from classical game theory that was pioneered by von Neumann [444] (see also the early work by Borel [86] and the book by von Neumann and

J. Rothe (ed.), *Economics and Computation*, Springer Texts in Business and Economics,
DOI 10.1007/978-3-662-47904-9_1

Morgenstern [445]), we will focus on algorithmic aspects of both noncooperative and cooperative games.

- *Computational social choice*, arising from classical social choice theory (see, e.g., the celebrated work of Nobel laureate Arrow [14]), is concerned with the computational aspects of voting. In Chapter 4, we will introduce a variety of voting systems and study their properties. Particular attention will be paid to algorithmic feasibility and complexity of winner determination, the related concepts of possible and necessary winners, and of various ways of influencing the outcome of elections by manipulation, control, or bribery. In addition, we will study the complexity of such manipulative attacks for so-called "single-peaked" electorates in Chapter 5. In Chapter 6, we will introduce to the emerging field of judgment aggregation where individual judgment sets of possibly interconnected logical propositions (rather than individual preferences as in voting) are aggregated to come to a collective decision.
- *Fair division*, finally, considers the problem of dividing goods among players who each can have quite different valuations of these goods. We will look at *"cake-cutting procedures"* in Chapter 7, aiming to divide one single, infinitely divisible good, the "cake." And, last but not least, we will study the issue of fairly allocating indivisible, nonshareable goods in Chapter 8, a field closely related to *"multiagent resource allocation"* and *"combinatorial auctions."* Concepts of fairness (such as envy-freeness) and social welfare optimization play a central role in these two chapters.

These three fields are closely interrelated, and we will highlight such cross connections throughout the book. In each of these areas, our particular focus is on the computational properties of the arising problems. Therefore, in Section 1.5, we will provide some background on algorithmics and complexity theory (and also on propositional logic) that will be useful in the subsequent chapters. Elementary basics of other mathematical fields (such as probability theory, topology, and graph theory) will be presented when they are needed, as tersely and informally as possible and with as many details as necessary. While all crucial concepts will be formally defined, most of them will be explained using examples and figures to make them easier to access. Moreover, many situations will be illustrated by short stories from everyday life, featuring Anna, Belle, Chris, David, Edgar (see Figure 1.1), and others.



(a) Anna (b) Belle (c) Chris (d) David (e) Edgar

Fig. 1.1 The main characters in this book

1.1 Playing

Part I of this book is concerned with algorithmic game theory, covering both noncooperative and cooperative games.

1.1.1 Noncooperative Game Theory

Smith and Wesson, two bank robbers, have been arrested. Since there is only little evidence that incriminates them and would stand up in court, they are offered a deal:

- If one of them makes a confession, he will be free to go (on probation), provided the other one remains silent, and this other one will then have to serve his ten years alone;
- if both confess, they both will be sent to prison for four years;
- but if both stubbornly keep silent, they can be sentenced only to two years' imprisonment (not for bank robbery, due to lack of evidence; only for minor offenses such as possession of unregistered weapons).

Unfortunately, they cannot coordinate their actions. What would be best for Smith to do in order to get away with a prison sentence as short as possible, given that its length does not depend only on his own, but also on Wesson's action? And what would be a most clever decision for Wesson, whose prison sentence conversely depends on Smith's action as well? That is the famous *"prisoners' dilemma"*!

George and Helena want to spend their first anniversary together and want to have some fun. George suggests to watch an exciting soccer game with his wife. Helena, however, would rather like to go to a concert with her husband. These are quite different things to do. Would it be better for each of them to try to enforce their own wish or to give way to their partner's wish? If none of them gives in, they won't spend their anniversary together! Every couple is well familiar with this *"battle of the sexes."*

Situations like that, where several individuals interact when making their decisions and where their gains depend also on the other individuals' decisions, can be described by strategic games. Almost a century ago, Borel [86] and von Neumann [444] wrote the first mathematical treatises in game theory. About twenty years later, the foundations of this theory as a standalone research area have been laid by von Neumann and Morgenstern in their groundbreaking work [445]. Evolving into a rich and central discipline within economics ever since, game theory has yielded many terrific insights and results, and a number of Nobel Prize winners in Economics, such as John Forbes Nash, Reinhard Selten, John Harsanyi, Lloyd S. Shapley, and Alvin E. Roth.

Basically, one distinguishes between cooperative and noncooperative games in this theory. The above examples are noncooperative games.

Noncooperative game theory, which we will be concerned with in Chapter 2, studies games where players face off against each other as lone, selfish fighters aiming to maximize their own gains. This category of games includes combinatorial games such as chess and go, but also card games such as poker, where the players have incomplete information about their opponents and where chance and the psychology of bluffing play an important role. However, not only board and gambling games can be expressed and studied in this theory, but all kinds of competitive situations (e.g., market strategies of companies or global strategies of states) can be modeled, too.

Some of the most central concepts in noncooperative games concern their stability, for example, with the intent to predict the outcome of such games. Is it possible for the individual players to choose their strategies so that they are all in equilibrium in the sense that no one has an incentive to deviate from the chosen strategy (provided all other players stick to their chosen strategies as well)? Historically, this question was of uttermost importance, for the entire human race, during the cold war between the Eastern bloc and the NATO countries, in light of the arms race and the doctrine of nuclear deterrence. One of the questions we study for noncooperative games is: How hard is it to find such equilibrium strategies?

1.1.2 Cooperative Game Theory

In cooperative game theory, which will be introduced in Chapter 3, we are concerned with players forming coalitions and working together, seeking to achieve their common goals. It may be possible to increase the gains of single players by cooperation with others. Whether a player joins a coalition or deviates from it certainly depends on whether or not this player will benefit from this.

Stability concepts for cooperative games are of quite central importance. If there is an incentive for a player to deviate from the grand coalition (the set of all participating players), then the game is instable and breaks up into several smaller coalitions working on their own and sometimes even competing with each other. In this chapter, we will see various notions that capture the stability of cooperative games in different ways. Also, one can measure the influence—or power—of a single player in such a game in various ways. Roughly speaking, one fundamental such power index of a player is based on how often this player's membership in a coalition is decisive for its success. Stability concepts and power indices in cooperative games will also be studied in terms of their algorithmic and complexity-theoretic properties where we will focus on games that can be compactly represented, such as games on graphs and weighted voting games.

1.2 Voting

Part II of this book is concerned with preference aggregation by voting—both in general and when restricted to single-peaked societies—and with judgment aggregation. In particular, we will focus on the social-choice-theoretic properties of voting systems and judgment aggregation procedures, and on the complexity of manipulative actions.

1.2.1 Preference Aggregation by Voting

Anna, Belle, and Chris meet for a joint evening of gaming. First, however, they have to agree on which game to play. Up for election are chess, poker, and Yahtzee.

"Let's play chess," Anna suggests. "And if that's not working for you, we might play Yahtzee. Poker I like the least."

"Oh no!" grumbles Chris. "Chess is *to-tal-ly* boring, and it's three of us."

"True, but we can just play a blitz chess tournament," replies Anna. "Then everyone will get to play."

"But Yahtzee is much more fun!" Chris disagrees. "And if you don't like that, we should at least play poker."

"I like poker the most," Belle pipes up now, "because I can bluff like hell, y'know. Yahtzee is what I like the least. If it's not poker, then we should play chess."

"Listen up!" says Anna now. "If we want to play, we need to agree on a game. But since each of us has different preferences, we should perhaps simply vote on what we will play."

Figure 1.2 shows the preferences of Anna, Belle, and Chris over these three alternatives, ordered from left (the most preferred alternative) to right (the least preferred one).

Before they can vote, however, they first need to agree on a voting system, a rule that says how to determine a winner from their individual preferences. There are a multitude of voting systems (many of which will be introduced in Section 4.1 starting on page 198). Democratic elections have been of central importance in human societies already since the roots of democracy have been planted in ancient Greece, and at least since the French Revolution and the Declaration of Independence of the United States of America. But, *which* rule should Anna, Belle, and Chris choose for their election?

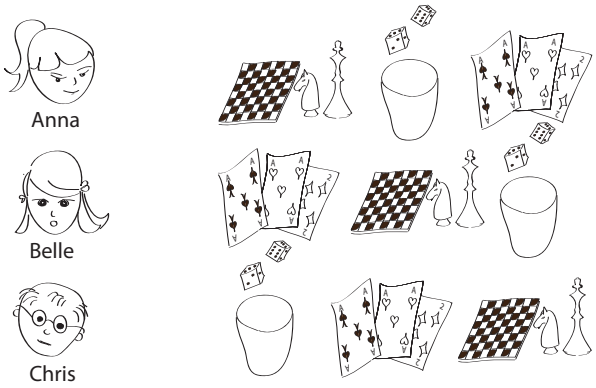


Fig. 1.2 Anna, Belle, and Chris are voting on which game to play

"All right then," Belle agrees, "so let us vote like that: Each two of the games go in a head-on-head contest . . . "

"I see," Chris interrupts. "Whatever game wins each of its pairwise comparisons, by which I mean it is preferred to each other game in at least two of our rankings, . . . "

". . . is the winner of the election and shall be played!" completes Anna.

"Exactly," says Belle. "Because then it must be better than every other game for us. After all, a majority of us prefer it to each other alternative."

The rule suggested by Belle was originally proposed by Marie Jean Antoine Nicolas de Caritat, the Marquis de Condorcet (1743–1794), a French philosopher, mathematician, and political scientist, in his essay from 230 years ago [163]. Condorcet's voting system is still very popular, since a Condorcet winner must be unique and there are good reasons indeed to consider a Condorcet winner the best alternative possible. However, Condorcet elections have their pitfalls as well.

"Great!" Anna bursts out. "Chess beats Yahtzee! Thanks to my and Belle's vote. Sorry, Chris, you can put away your dice box now and . . . "

"Not so fast, please!" Belle interrupts her. "No frigging way are we playing chess! Poker beats chess due to my and Chris's vote, so . . . as likely as not will we play poker." She thinks. "After all, if Yahtzee is beaten by chess and chess is beaten by poker, then poker must beat Yahtzee, too, must it not?"

"Not so fast, girls!" Chris pipes up now. "Screw poker! Because Yahtzee beats poker thanks to Anna's and my vote!"

The three look at each other, bewildered. "What the ... I mean, what *game* has won now?" they ask themselves in unison.

The winner is: ... *none* of the three games! This feature of the Condorcet voting system is known as the *Condorcet paradox*: A *Condorcet winner* (i.e., a candidate who beats every other candidate in pairwise comparison) does not always exist! Even though the individual rankings of the three voters are rational in the sense that there are no cycles, the societal ranking produced by the Condorcet rule is cyclic: Poker beats chess in their head-on-head contest, chess beats Yahtzee, and yet Yahtzee beats poker. This *Condorcet cycle* is depicted in Figure 1.3.

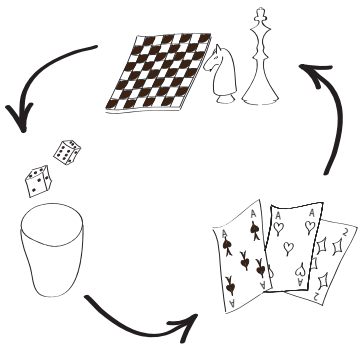


Fig. 1.3 The Condorcet paradox

Since the work of Condorcet [163], social choice theory has been concerned with collective decision making by aggregating individual preferences via voting. Chapter 4 provides the foundations of this theory and introduces a variety of voting systems and their properties. In social choice theory, too, path-breaking insights have been honored by Nobel Prizes in Economics, given to Kenneth Arrow and John Hicks for their pioneering contributions to general economic equilibrium theory and welfare theory (see, for example, Arrow's famous impossibility theorem [14], stated here as Theorem 4.1 on page 237) and to Amartya Sen for his contributions to welfare economics.

Not only the players in a game, but also the voters in an election can act strategically, by reporting a vote that they consider more useful for their goal than their true preferences and that they hope will—depending on the voting system and on the other voters' preferences—ensure their favorite candidate's victory. By the celebrated Gibbard–Satterthwaite theorem [521]

(see Theorem 4.2 on page 239), no reasonable voting system is protected against this kind of manipulation. Again, we will focus on the algorithmic and complexity-theoretic aspects: How hard is it to know if one can set one's individual preferences strategically so as to successfully manipulate the election? Also other ways of influencing the outcome of elections, such as electoral control (e.g., by adding or deleting either candidates or voters) and bribery, will be introduced and thoroughly discussed in Chapter 4.

1.2.2 Manipulative Actions in Single-Peaked Societies

Relatedly, in Chapter 5 we will study the complexity of control, manipulation, and bribery problems when restricted to so-called *single-peaked electorates*. This notion has been introduced by Black [77, 78], and is now considered to be "*the* canonical setting for models of political institutions" [295, p. 336]. Single-peaked electorates can be used to model societies that are heavily focused on a single issue (such as taxes, public budgets, cutbacks in military expenditure, etc.) where political positions can be ordered on an axis (or, in a "left-right spectrum").

1.2.3 Judgment Aggregation

In Chapter 6, we will turn to a field called *"judgment aggregation,"* which is much younger than the related field of preference aggregation by voting. Unlike in voting, the individual judgments of experts (the *"judges"*) regarding propositions (that can be logically connected) are to be aggregated here. In judgment aggregation, too, very interesting paradoxical situations may occur, such as the so-called *doctrinal paradox* (relatedly, the *discursive dilemma* [375, 473]), which will be explained in more detail in this chapter.

Just as in elections, it is here possible to influence the outcome of a judgment aggregation procedure. External actors might try to obtain their desired collective judgment set from the experts by bribing them. The judges themselves might try to manipulate the outcome by reporting insincere individual judgments instead of their true ones. An external expert might try to influence the outcome by controlling the structure of the used judgment aggregation procedure, e.g., by adding or deleting judges. The investigation of the algorithmic and complexity-theoretic properties of the related problems has been initiated by Endriss, Grandi, and Porello [236] only recently and has opened a currently very active field of research, which will also be surveyed in this chapter.

1.3 Dividing

Part III of this book is concerned with fair division of either divisible or indivisible goods.

1.3.1 Cake-cutting: Fair Division of Divisible Goods

"Ein Kompromiss," German economist and politician Ludwig Erhard (Chancellor of West Germany from 1963 until 1966) is quoted as saying, *"das ist die Kunst, einen Kuchen so zu teilen, dass jeder meint, er habe das größte Stück bekommen."* In English: "A compromise is the art of dividing a cake in such a way that everyone believes he has the biggest piece."

We will be concerned with the fair division of cake (or, *"cake-cutting"*) in Chapter 7. The "cake" is only a metaphor here that can be applied to any divisible resource or good.

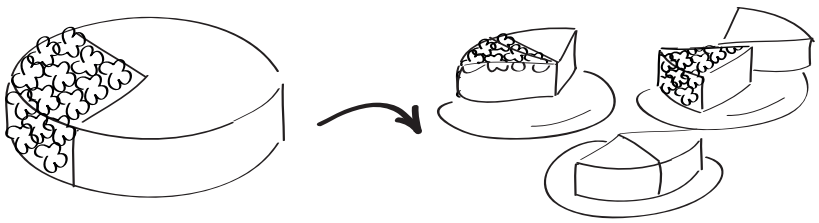


Fig. 1.4 Division of a cake into three portions

Figure 1.4 shows a division of a cake into three portions. But when can it be called a "fair" division? How many parents have failed miserably when trying to fairly divide a cake among their children? In the worst case, *all* the children believe to have received just the worst of all pieces and feel all hard done-by. Would that be easier perhaps if the children themselves were to cut the cake and divide it among them? Not very likely: Arguments and fights will start as soon as envy raises its ugly head. On a larger scale, how many mediators have failed miserably when trying to fairly divide debatable territories among opposing parties, be it in the Middle East, Central Africa, the Balkans, in East Europe, or in other parts of the world?

The purpose of cake-cutting protocols is to produce a fair division of the cake and to make all participating players as happy as possible. "Fairness" can be interpreted differently. It can mean, for instance, that no player receives a smaller share of the cake than is due to them with respect to their own individual valuation, i.e., everyone gets at least a proportional share. Or, it can also mean that there is no envy among the players. If possible,

the protocol should even *guarantee* these properties. That is, proportionality or even envy-freeness of the allocation should be achieved independently of the players' individual valuations. Steinhaus [559] was one of the first to formulate the problem of fair division in cake-cutting as a most challenging and beautiful mathematical problem and to propose first solutions to it. In this chapter, we will survey much of the work that has been done in this field since his groundbreaking paper, focusing on valuation criteria such as various notions of fairness and introducing many concrete cake-cutting protocols.

1.3.2 Fair Division of Indivisible Goods

In Chapter 8, finally, we will be concerned with a problem that is closely related to cake-cutting, namely the problem of fair division of *indivisible, nonshareable* goods or resources. All participating agents (or, players) have individual, subjective (ordinal or cardinal) preferences over the single goods, or all bundles of goods. We will classify the resulting allocation problems and, distinguishing between ordinal preferences and cardinal utilities, will first focus on the problems of preference elicitation and compact representation, and will then describe ordinal and cardinal fairness criteria. Next we will turn to centralized fair division mechanisms for computing allocations, again distinguishing between ordinal and cardinal preferences, and will then describe a number of decentralized fair division mechanisms. Finally, we will briefly discuss a number of further issues in this chapter, such as strategy-proofness and matching.

As Chapter 8 focuses on *fair division*, its authors decided to not cover the related area of *(combinatorial) auctions* with its many applications, for example in e-commerce. Auctions will therefore be under-covered with respect to their importance in the literature (see the textbook *"Combinatorial Auctions"* edited by Cramton, Shoham, and Steinberg [177]) in this book.

In addition to the individual utility each player can realize for herself, one is also interested in optimizing the *societal* utility, the "social welfare," which can be measured in various ways (including utilitarian and egalitarian social welfare, see Section 8.4.2 starting on page 511). Finding an optimal allocation of all goods that maximizes social welfare is a difficult combinatorial problem whose algorithmic and complexity-theoretic properties will briefly be surveyed in Section 8.5.2 starting on page 522. However, as mentioned above, combinatorial auctions will *not* be treated in this chapter, a field much too vast for exposing it in detail on just a few pages. Instead, we point the reader to the above-mentioned book edited by Cramton, Shoham, and Steinberg [177] (see also the books by Shoham and Leyton-Brown [542] and Wooldridge [607]).

On the other hand, it would be unthinkable not to address auctions in a book on economics and computation. Therefore, in the following section we

will briefly digress to informally introduce a number of common *single-item auctions* where the single objects are auctioned off to the bidders—as agents are called in the field of auctions—one by one (by contrast, in a *combinatorial auction*, the bidders make their bids for *bundles* of objects).

1.3.3 A Brief Digression to Single-Item Auctions

Every bidder knows his individual value for each of the single objects. However, he of course is not obliged to truthfully announce his actual values for the objects during the bidding phase, but he may bid quite different prices for the objects. The bidder's goal is to win each desired object for a price as low as possible. The auctioneer's goal, however, is to make as much profit as possible from auctioning off all items. In this sense, bidders and auctioneer are adversaries in this "auction game" and, in addition, the bidders compete with each other.

1.3.3.1 Classification

The bidders can make their bids

- either *openly*—in so-called *open-cry auctions*, i.e., all bidders know which bids have been made so far (possibly not by whom, though),
- or *concealed*—in so-called *sealed-bid auctions*, i.e., no bidder knows any of the previously made bids of other bidders.

Open-cry auctions can be further subdivided into

- *ascending-price auctions* where the bids go from lower to higher values, and
- *descending-price auctions* where the bids go from higher to lower values.

Winner determination is an important issue in an auction: Which bidder wins which object and what does she have to pay for it? One can distinguish, for example, between

- *first-price auctions* where a bidder wins an object by making the highest bid and has to pay this price, and
- *second-price auctions* where, again, a bidder making the highest bid wins the corresponding object, but has to pay only the price of the second-highest bid.

We now introduce a number of "classical" types of single-item auctions. The above classification criteria can be combined with each other. For example, one thus obtains a *first-price, sealed-bid auction*: There is only a single round in which the bidders hand in an envelope containing their bid, and

whoever has made the highest bid wins this object (if need be, ties are broken according to some preassigned rule). The winner pays the amount corresponding to his bid. This type of auction is often used, for example, in public auctions of real estate property.

What would be a good strategy for a bidder in such an auction? The highest bid wins, but the greater the difference is between the highest and the second-highest bid, the more money has been wasted by the winner. Making a bid just a tiny bit above the second-highest bid would have been enough to ensure victory, and would have saved a lot of money. Therefore, the best strategy is to make a bid that is lower than one's true value for this object. How much lower, though, depends on the circumstances—for example, it depends on what the unknown bids of the other bidders are, and on how desperately one desires to get just this object. There is no general solution to this problem; some risk is unavoidable. Auctions are like bets and how risk-loving a bidder is will influence his possible wins and losses.

1.3.3.2 English Auction

This is an ascending-price, first-price, open-cry auction. At the beginning, the auctioneer announces a minimum price (which may also be 0) for the object on the table and waits for higher bids. The bidders know the current bid, which must be exceeded by each new bid. If no bidder raises the current bid, the object goes to whoever made the last bid for the price of his final bid. If none of the bidders raises the initial minimum price, the auctioneer gets the object for this minimum price. Famous auction houses like *Sotheby's* (originally situated in London but now headquartered in New York City) and *Christie's* (with its main headquarters also located in London and New York City) conduct this type of English auction.

What would be a good strategy for a bidder in such an auction? The highest bid wins again, and the winner again wastes the more money, the greater the difference is between the highest and the second-highest bid. Therefore, it's in the interest of the bidder to keep this difference as small as possible. Since bids are made openly, it would be smart to raise the current bid only by a small amount. This approach, though, is based on the assumption that all bidders behave rationally. However, auctions are subject to psychological aspects, which are not that simple to be modeled game-theoretically. For example, it might be that two bidders work themselves up into fierce fighting, taking turns in outbidding each other again and again, and eventually the winner may have paid a lot more money than he would have paid if only he had raised—substantially!—the first bid just once, right at the beginning of the auction, in order to frustrate his rival. Would this strategy have worked better? Of course, that's also just a speculation.

Wooldridge [607] highlights another interesting feature of English (and other) auctions: If the true value of the object is unknown or uncertain, the

"winner's curse" may occur, not only in English auctions, but particularly often in these. This refers to the tendency of the auction winner to overpay an object. For example, if there is an auction for a house that has not been thoroughly evaluated by experts, it is unclear if a bidder who wins this house in an auction should really be happy, or if he rather should be worried to have paid too much for it. Perhaps the other bidders stopped bidding only because they knew more than him about the constructional condition of the house.

1.3.3.3 Dutch Auction

This is a descending-price, first-price, open-cry auction (and is also called *clock auction*). At the beginning, the auctioneer announces an obviously too high price for the object on the table, which lies above the expected maximum price of the bidders. Then the auctioneer lowers the price step by step until there is a bidder who is willing to accept and pay the current price. This bidder receives the object for this price.

What would be a good strategy for a bidder in such an auction? It wouldn't make sense for a bidder to accept a price above her own, true value for this object. Once the auctioneer's offered price drops below this true value, the bidder's potential gain increases the longer she waits to accept a current price, but at the same time her risk also increases to lose the object to somebody else. As in the English auction, the *winner's curse* can occur here as well, namely if a bidder gets cold feet too early and accepts the current price, although it is only a tiny little bit lower than her true value for the object. This is why Dutch auctions tend to end pretty quickly.

1.3.3.4 Vickrey Auction

A *Vickrey auction*—named after its inventor William Vickrey, who in 1996 received the *Nobel Prize in Economics* jointly with James Mirrlees—is a second-price, sealed-bid auction. As in a first-price, sealed-bid auction, there is only one round where the bidders submit their bids in a sealed envelope. The winner is again a bidder with the highest bid, where a tie-breaking rule is applied to break ties, if there occur any. However, the winner does not have to pay the highest, but only the second-highest price. This may not seem intuitively sensible at first, but it does have a big advantage, as we will now see.

What would be a good strategy for a bidder in such an auction? The above-mentioned advantage of Vickrey auctions is that telling the truth—i.e., bidding their true value of the object—is the dominant bidder strategy (in the sense of Definition 2.2 on page 46). To understand why this is so, let's

have a look at the following two cases, where we assume that the second-highest bid remains the same:

Case 1: The bidder bids more than her true value. Then it is more likely than when telling the truth for the bidder to get her bid accepted and win the object. If she really wins it, however, she may have to pay more than what she would have paid had she remained truthfully. In other words, by being dishonest the bidder only increases her chances to make a loss.

Case 2: The bidder bids less than her true value. Then it is less likely than when telling the truth for the bidder to get her bid accepted and win the object. However, even if she does win it, the price she has to pay has not been influenced by her bidding less than her true value: She still has to pay the second-highest price. That is, the bidder does not have an advantage in terms of gain maximization; all she has achieved by being dishonest is to lower her winning chances.

Since there is no advantage for the bidder to deviate from her true value in either way, it can be expected that she will make a truthful bid. Auctions similar to Vickrey auctions are used, for example, on the internet auction platform eBay. The *Vickrey–Clarke–Groves mechanism* generalizes Vickrey auctions from single-item to combinatorial auctions (see, e.g., the books by Cramton, Shoham, and Steinberg [177], Shoham and Leyton-Brown [542], and Wooldridge [607]).

1.3.3.5 American Auction

This is a special type of auction that is often used in charity events. In our classification scheme, it is an ascending-price, open-cry auction; however, in contrast with the English auction, the winner of an object does not pay the highest price, and does not pay the second-highest price either, but instead *each* bidder, immediately when making a new bid, has to pay the difference between his bid and the previous bid. The auctioneer may predetermine the allowed difference amounts between bids, and there is a variant in which the auctioneer also fixes the maximum duration of the auction, which is not known to the bidders. In this variant of the American auction, the last bid made prior to termination is accepted and wins.

What would be a good strategy for a bidder in such an auction? On the one hand, compared with, say, English auctions, American auctions increase the potential gain of the bidders, since a lucky bidder can win a valuable object with just one bid—the last one—for really little invested money (namely, merely the difference to the second-to-last bid). On the other hand, all bidders not winning the object come away empty-handed, that is, they lose all their invested money and might make a really big loss. In fact, the actual winner of an American auction is the auctioneer, who typically makes quite a lot more

money than, e.g., in an English auction. One reason why American auctions are popular in charity events is that also the losers can better put up with their losses, knowing that their money has been gambled away for a good purpose. American auctions resemble gambling more than other auctions.

In an American auction with prefixed maximum duration, the winning decision is made only close to its end. Therefore, a good bidder strategy is to make bids only in the final phase. The problem, of course, is that the maximum length is unknown to the bidders, so they must guess when it is a good point in time to get in with their own bids. Moreover, when there are several objects to be auctioned off, one has to give consideration to which objects one would be most interested in bidding on. If one is too eager to bid on each object right from the start and then stops too early so as to not lose too much money, one is likely to maximize one's loss only.

Another variant of the American auction is known as *first-price, sealed-bid, all-pay auction*. In this variant, all bidders submit their bid in a sealed envelope, and the bidder with the highest bid wins (again making use of a tie-breaking rule if needed), but *all* bidders have to pay the price corresponding to their bid. In this way the auctioneer would carry his own gain to an extreme. However, he normally wouldn't find bidders willing to go into such an auction with their eyes open, since these rules are commonly considered to be rather unfair. That is why this quite special form of an auction is rather of theoretical interest; for example, the economic effect of lobbyism, party donations, and bribery can be modeled this way (cf. Section 4.3.5). Considering donators as bidders and parties as sellers of a "good" (namely political influence), the bidder with the highest donation wins in this model (assuming that this donation has helped the party to win the election) and increases his political influence. The other bidders come away empty-handed and increase their political influence not at all or only marginally, but their donations will still not be refunded (for details, see [5, pp. 83–84]).

1.3.3.6 Expected Revenue

The example of the American auction shows that an auctioneer has to be concerned about how to maximize the total revenue of an auction whenever possible. This concerns in particular choosing the right auction protocol, and different perspectives of the parties involved have to be taken into account. For example, American auctions are preferred from the point of view of an auctioneer, though they are unlikely to be accepted by the bidders. Sandholm [518, p. 214] discusses this issue (see also [607, pp. 297–298]) and comes to the following result for the other types of auction considered above (first-price, sealed-bid auction; English auction; Dutch auction; and Vickrey auction). In particular, the answer to the question raised above very much depends on both the auctioneer's and the bidders' risk-taking propensity:

- For *risk-neutral bidders*, the auctioneer's expected revenue is provably identical for these four auction types (under certain simple assumptions). That is, the auctioneer can expect the same revenue on average, no matter which type of auction he chooses among these four.
- Maximizing the gain is not the most important issue for a *risk-loving bidder*, i.e., such a bidder would rather like to indeed receive an object, even if that means to pay something more for it. In this case, Dutch auctions and first-price, sealed-bid auctions provide the auctioneer with a higher expected revenue. That is so because risk-loving bidders may be inclined to increase their winning chances at the expense of their own profit by raising their bids a bit higher than risk-neutral bidders.
- By contrast, a *risk-averse auctioneer* would be better off on average when using English or Vickrey auctions.

These assertions are to be treated with caution, though. For example, the first statement critically depends on the bidders really having *private* values for the objects, known only to themselves.

1.4 Some Literature Pointers

In the single chapters about playing (Part I), voting and judging (Part II), and dividing (Part III), we will often focus on algorithmic aspects. In the central Part II, which with three chapters is a bit more extensive than the other two parts, we are dealing with the problems of *computational social choice* (COMSOC). On the one hand, COMSOC applies methods of computer science (such as algorithm design, complexity analyses, etc.) to social choice mechanisms (such as voting systems or judgment aggregation procedures). On the other hand, concepts and ideas from social choice theory are integrated into computer science, for example, in the area of distributed artificial intelligence and, most notably, in the design of multiagent systems, networks, ranking algorithms, recommender systems, and others.

Since 2006, scientists from all over the world, who work in these areas ranging from economics and the political sciences to computer science, meet biennially at the *International Workshop on Computational Social Choice*, which so far took place at the universities of Amsterdam, The Netherlands (2006), Liverpool, UK (2008), Düsseldorf, Germany (2010), Kraków, Poland (2012), and at Carnegie Mellon University in Pittsburgh, USA (2014). The (informal) proceedings of these workshops have been edited by Endriss and Lang [239], Endriss and Goldberg [233], Conitzer and Rothe [167], Brandt and Faliszewski [126], and Procaccia and Walsh [484]. If you like reading this book, you will also enjoy reading these proceedings, to get deeper insights into the challenging research questions in this fascinating area.

Moreover, Brandt et al. are currently editing the *"Handbook of Computational Social Choice"* [125] to be published by Cambridge University Press.

There are also many other, older textbooks on algorithmic game theory (e.g., the textbook *"Algorithmic Game Theory"* edited by Nisan et al. [456]), social choice theory and welfare economics (e.g., the two volumes of the *"Handbook of Social Choice and Welfare"* edited by Arrow, Sen, and Suzumura [15, 16]), and fair division (e.g., the books by Brams and Taylor [117] and Moulin [432]), and some more will be mentioned in the relevant chapters of this book.

The reader is also referred to the surveys by Chevaleyre et al. [152, 156], Conitzer [164], Daskalakis et al. [190], Faliszewski, Hemaspaandra, and Hemaspaandra [261], Faliszewski and Procaccia [274], Nguyen, Roos, and Rothe [450], and Rothe and Schend [512], as well as to the book chapters by Brandt, Conitzer, and Endriss [124], Faliszewski et al. [268], and Baumeister et al. [58]. They each cover some more specific topics in computational social choice, algorithmic game theory, and fair division.

But now, let's get ready to play, vote, and divide ...

Stop! Before we can actually start playing, voting, and dividing, a few more preliminaries are needed.

1.5 A Brief Digression to Computational Complexity

As mentioned earlier, we will often focus on the algorithmic and complexity-theoretic aspects of problems related to playing, voting, and dividing in this book. But, how exactly does one determine a problem's algorithmic or complexity-theoretic properties? To answer this question, we will now briefly digress, introducing to the foundations of complexity theory as terse and informally as possible and in as much detail as necessary. Crucial concepts of this theory, which will be important in almost all the following chapters, will be illustrated for SAT, the satisfiability problem of propositional logic. One can also skip this digression for now and, only when needed, come back later to look up one notion or another.

1.5.1 Some Foundations of Complexity Theory

For about half a century now, problems have been investigated and classified in terms of their computational complexity. Having developed from computability theory (embracing, in particular, recursive function theory) and the theory of formal languages and automata (see, e.g., the textbooks by Homer and Selman [340], Hopcroft, Motwani, and Ullman [341], and Rogers [504]), complexity theory is a subarea of theoretical computer science. Among other questions, computability theory studies which problems are algorithmically

solvable and which are not; the latter are so-called *undecidable problems*, such as the so-called *"halting problem for Turing machines."* By contrast, complexity theory is concerned with algorithmically solvable (i.e., *"decidable"*) problems only, but specifically asks for the computational costs required to solve such problems. There are many useful textbooks and monographs on computational complexity, such as those by Bovet and Crescenzi [102], Hemaspaandra and Ogihara [335], Papadimitriou [462], Rothe [508], Wagner and Wechsung [587], Wechsung [595], and Wegener [596]. Algorithmic and complexity-theoretic concepts and methods for problems related to playing, voting, and dividing will be important in many chapters of this book.

1.5.1.1 Turing Machines and Complexity Measures

Complexity theory is to algorithmics as yin is to yang.[1] While the main objective of algorithmics is the design of algorithms that are as efficient as possible and thus provide as low *upper* (time) bounds as possible for solving a given problem, in complexity theory one tries to prove that certain problems cannot be solved efficiently at all, that is, those problems do not have any efficient algorithm whatsoever. In other words, the goal of complexity theory is to prove that solving these problems algorithmically requires as high a *lower* (time) bound as possible. And once the least upper and greatest lower bounds of a problem meet, its complexity analyst and algorithm designer (who often are just one and the same person) feels perfect harmony and enthusiastically calls this problem "classified."

Here, *computation time* (or, *runtime*) is a discrete complexity measure defined as the number of elementary computation steps an algorithm performs as a function of the size of the input. The input size depends on the chosen encoding of problem instances. Usually, these are encoded in binary, i.e., as a string over the aphabet $\Sigma = \{0,1\}$, so that they can be processed by a computer executing the algorithm. Different encodings usually have only a negligible influence on the computation time of an algorithm; note that constant factors are commonly neglected in the runtime analysis of algorithms. Therefore, we will disregard coding details.[2] What is meant by an "elementary computation step" of an algorithm depends on the algorithm model at hand. In theoretical computer science, and particularly so in complexity theory, it

[1] In Chinese philosphy, *yin and yang* denote opposite or contrary forces that are mutually dependent and interconnected: Opposites that give rise to each other by interrelating to one another and that cannot exist without each other.

[2] That, however, is not to say that we will disregard *representation details*. In fact, for many of the problems we will study it is utterly important to choose an appropriate, compact representation to be able to handle them algorithmically. This refers, for example, to Section 2.5.2 about how to represent problems related to compute equilibria of (noncooperative) games, to Section 3.2.2 about the representation of simple (cooperative) games, and to Section 8.3 about preference elicitation for and compact representation of allocation problems for indivisible goods.

is common to use the model of *Turing machine*, named after its inventor Alan Turing [579]. This is a very simple model, and yet it is universal: By Church's thesis [162], everything computable is computable on a Turing machine. Again, we omit formal details here and instead refer to the literature on computability and complexity theory given above. All algorithms in this book will be described only informally; whether they are implemented on an abstract algorithm model such as a Turing machine or in some concrete programming language does not matter for our purposes. We will distinguish, though, between various types of Turing machines or algorithms (where we confine ourselves to *"acceptors,"* suitable to solve decision problems only):

- In the computation of a *deterministic Turing machine* on any input, each computation step is uniquely determined, i.e., for each current *"configuration"* of the Turing machine (which is a complete *"instantaneous description"* of the computation at this point in time), there is a unique successor configuration until, eventually, an *end configuration* (or *halting configuration*) is reached that is either *accepting* or *rejecting*. Note that it is also possible, in principle, that a computation never terminates, i.e., it never reaches an end configuration. This is important in computability theory and is closely related to the halting problem's undecidability.[3]
- The computation of a *nondeterministic Turing machine* on any input is more general: In each computation step (before reaching an end configuration), it is possible to branch nondeterministically, i.e., at each point in time during the computation, the current configuration can have more than one successor configuration until, eventually, an accepting or rejecting end configuration is reached. Therefore, a nondeterministic computation is not a deterministic sequence of configurations, but instead we have a *nondeterministic computation tree*
  - whose root is the start configuration (which, in particular, encodes the input string),
  - whose inner vertices are the configurations reachable from the start configuration before the computation has terminated, and
  - whose leaves are the accepting and rejecting end configurations.

  For an input to be accepted it is enough that there exists at least one accepting computation path in this computation tree. An input is rejected only if all paths of this tree lead to rejecting end configurations.

Again, note that the computation tree of a nondeterministic Turing machine running on some input can have infinite computation paths, in principle. However, since we will be concerned with *decidable* problems only and since

[3] Roughly speaking, the *halting problem (for Turing machines)* is the following: Given (an encoding of) a Turing machine and an input, does the machine on that input ever halt? That this famous problem is undecidable means that there is no algorithm that can solve this problem.

for those it is possible to "clock" (deterministic and nondeterministic) Turing machines, we can safely ignore the possibility of infinite computations.

In addition to determinism and nondeterminism there are many other computational paradigms (for example, randomized algorithms), and in addition to computation time there are many other complexity measures (such as the *computation space*, or *memory*, required to solve the problem at hand). However, these will only rarely be considered here (namely, for PSPACE); we will mostly restrict ourselves to analyzing the computation time of (deterministic or nondeterministic) algorithms or to prove lower time bounds of problems.

1.5.1.2 The Complexity Classes P and NP

In complexity theory, one collects all those problems whose solutions require roughly the same computational cost with respect to some complexity measure (e.g., computation time) in so-called *complexity classes*, and the most central time classes are:

- P (*"deterministic polynomial time"*) and
- NP (*"nondeterministic polynomial time"*).

P (respectively, NP) is defined as the class of problems that can be solved by some deterministic (respectively, nondeterministic) Turing machine in polynomial time. Deterministic polynomial-time algorithms are thought of as being efficient, since a polynomial, such as $p(n) = n^2 + 13 \cdot n + 7$, typically grows relatively moderately, as opposed to the explosive growth of an exponential function, such as $e(n) = 2^n$. For very small input sizes n, it may be true that function e has rather benign values (for example, we have $e(n) = 2^n < n^2 + 13 \cdot n + 7 = p(n)$ for each $n < 8$), but already for slightly larger input sizes, the exponential function blows up compared with the polynomial (e.g., if $n = 30$, note that $e(30) = 1,073,741,824$ is considerably greater than $p(30) = 1,297$). For even larger input sizes (say, for $n = 100$), that are not uncommon in practice, exponential functions can reach nothing less than astronomically large values. For example, the number of all atoms in the visible universe (dark matter excluded) has been estimated to be roughly 10^{77}. Suppose that an algorithm having a runtime of, say, 10^n and destroying one atom per computation step, runs on an input of size 77. When this algorithm terminates, it will have wiped out the entire visible universe, including itself[4] (yes, this is a visual algorithm and so belongs to the *visible* universe). However, even for an algorithm that does not destroy atoms during its computation (which is the normal case for algorithms), whoever has started it to run on that input will have passed away a long, long time before the algorithm finally comes up with its answer.[5]

[4] Which is why the assumption it would ever terminate is absurd.

[5] Like "42" as per Adams [2], although it is not quite clear in this case what the input was 7.5 million years before supercomputer Deep Thought came up with this answer.

In contrast, *non*deterministic polynomial-time algorithms are viewed as not being efficient in general. If one would try to simulate an NP algorithm deterministically (which seemingly would amount to checking each path in the nondeterministic computation tree, one by one, systematically searching for an accepting end configuration), this deterministic algorithm would require exponential time. Namely, if the runtime of an NP algorithm running on inputs of size n is bounded by a polynomial $p(n)$ and if we assume that the NP algorithm, on any input, branches in each inner vertex of its computation tree into at most two successor vertices, then there can be up to $2^{p(n)}$ paths in the computation tree, and only after the last one of these paths has been checked without success, one can be sure that there really is no accepting computation path. Of course, this is not a proof that P is not equal to NP.

Indeed, the "P = NP?" question has been open since more than 40 years now. This is perhaps the most important open problem in all of theoretical computer science and it is one of the seven *millennium problems* whose solutions will each be rewarded with a prize money of one million US dollars by the Clay Mathematics Institute in Cambridge, Massachusetts. It is clear that $\mathrm{P} \subseteq \mathrm{NP}$; the challenging open question is whether or not this inclusion is strict. In 2002, Gasarch [300] conducted a "P =? NP poll" among complexity theoreticians where the overwhelming majority indicated by their vote that they believe that $\mathrm{P} \neq \mathrm{NP}$. However, no one succeeded so far with providing an actual *proof* of this widely believed inequality; and whenever a "proof" for it has spread in the past, it didn't take long to detect its flaws. The prize money of one million US dollars for a *correct* solution to this fascinating open problem is still out there waiting for you to pocket it!

1.5.1.3 Upper and Lower Bounds

In order to show an *upper bound* $t(n)$ on the (time) complexity of a problem, it is enough to find some *specific* algorithm solving this problem in time t, i.e., it works for all inputs of size n in time at most $t(n)$. A P algorithm, for instance, shows that the corresponding problem can be solved in time $p(n)$ for some polynomial p and, therefore, is considered to be efficiently solvable. As mentioned earlier, when stating an upper bound we can neglect constant factors and also finitely many exceptions, as we are interested only in the *asymptotic rate of growth* of complexity functions.

The following notation describing the asymptotic rates of growth of functions (by orders of magnitude) are due to Bachmann [33] and Landau [382].

Definition 1.1 (asymptotic rates of growth). Let s and t be functions from $\mathbb{N}$ into $\mathbb{N}$, where $\mathbb{N}$ is the set of nonnegative integers.

1. $s \in \mathcal{O}(t)$ if and only if there exist a real constant $c > 0$ and an integer $n_0 \in \mathbb{N}$ such that $s(n) \leq c \cdot t(n)$ for all $n \in \mathbb{N}$ with $n \geq n_0$.
   If $s \in \mathcal{O}(t)$, we say that s *asymptotically does not grow faster than* t.

2. $s \in \Omega(t)$ if and only if $t \in \mathcal{O}(s)$.
   If $s \in \Omega(t)$, we say that s *asymptotically grows at least as fast as* t.
3. The class $\Theta(t) = \mathcal{O}(t) \cap \Omega(t)$ contains all functions that have the same *asymptotic rate of growth as* t.
4. $s \in o(t)$ if and only if for all real constants $c > 0$, there exists an integer $n_0 \in \mathbb{N}$ such that $s(n) < c \cdot t(n)$ for all $n \in \mathbb{N}$ with $n \geq n_0$.
   If $s \in o(t)$, we say that s *asymptotically grows strictly slower than* t.
5. $s \in \omega(t)$ if and only if $t \in o(s)$.
   If $s \in \omega(t)$, we say that s *asymptotically grows strictly faster than* t.

For example, $p \in \mathcal{O}(e)$ (and thus also $e \in \Omega(p)$) for the exponential function $e(n) = 2^n$ and the polynomial $p(n) = n^2 + 13 \cdot n + 7$ defined above. Since every exponential function asymptotically not only grows at least as fast as, but even strictly faster than each polynomial, we even have $p \in o(e)$ (and $e \in \omega(p)$). It is also clear that $\mathcal{O}(1)$ is the class of all constant functions. The crucial difference between the definitions of the $\mathcal{O}$ and o notation is the quantifier over the real positive constants c. While the existential quantifier before c in the definition of $\mathcal{O}(t)$ ensures that one may neglect arbitrarily large constant factors, the universal quantifier before c in the definition of $o(t)$ achieves that $c \cdot t(n)$ is greater than $s(n)$ even if the growth of $c \cdot t(n)$ is slowed down by an arbitrarily small constant factor c—that much faster grows t in comparison with s (see also, e.g., [508, 315] for further details, properties, and examples).

Upper bounds for the complexity of a problem are given in the $\mathcal{O}$ notation, and it is enough to find a suitable algorithm solving the problem within the time allowed by this upper bound. By contrast, a *lower bound* $t(n)$ on the (time) complexity of a problem means that *no* algorithm whatsoever can solve this problem in less time than allowed by $t(n)$. At least time $t(n)$ is required to solve this problem for inputs of size n (perhaps not for all of them, but at least for some inputs of this size), and no matter which algorithm of the considered algorithm type we choose. Again, we neglect constant factors and allow finitely many exceptions.

Obviously, both aspects—the upper and the lower time bounds for solving a problem algorithmically—are closely related, they are two sides of the same coin. If one succeeds in finding even *matching* (asymptotic) upper and lower bounds for some problem, one has determined its inherent complexity, at least with respect to the considered class of algorithms.[6] Unfortunately, that is not very often possible.

[6] To avoid misunderstandings: Finding matching upper and lower bounds for a problem does not mean to find an algorithm with running time $\Theta(t)$ that solves the problem. That would merely show that the running time of *this particular* algorithm (which provides *some* upper bound only) has been analyzed well, so that no essential improvements are possible in the runtime analysis of this algorithm. By constrast, to show a lower bound matching this upper bound of $\Theta(t)$, one would have to show that *no* algorithm (of the considered type) for this problem has a runtime that asymptotically is better than that.

1.5.2 The Satisfiability Problem of Propositional Logic

In this section, we will introduce SAT, the famous satisfiability problem of propositional logic, discuss its upper bounds and how to improve them, and then turn to how to prove lower bounds for computational problems such as SAT in general, provding some basic tools from complexity theory. Finally, we give some background on approximation theory.

1.5.2.1 Definitions

Consider, for example, the famous *satisfiability problem* of propositional logic, denoted by SATISFIABILITY (or, shorter, by SAT). Decision problems like that, whose question is to be answered by "yes" or "no," will be represented in the following form:

SATISFIABILITY (SAT)	
Given:	A boolean formula φ.
Question:	Is there a satisfying truth assignment to the variables of φ, i.e., an assignment that makes φ evaluate to true?

Here, a *boolean* (or, *propositional*) *formula* consists of *atomic propositions* (also called the *variables of the formula*) connected by *boolean operations*, such as the following:

- *conjunction* (i.e., *AND* or, symbolically, $\wedge$),
- *disjunction* (i.e., *OR* or, symbolically, $\vee$),
- *negation* (i.e., *NOT* or, symbolically, $\neg$),
- *implication* (i.e., *IF ... THEN ...* or, symbolically, $\Longrightarrow$),
- *equivalence* (i.e., *... IF AND ONLY IF ...* or, symbolically, $\Longleftrightarrow$),
- etc.

Table 1.1 Truth table for some boolean operations

x	y	$x \wedge y$	$x \vee y$	$\neg x$	$x \Longrightarrow y$	$x \Longleftrightarrow y$
0	0	0	0	1	1	1
0	1	0	1	1	1	0
1	0	0	1	0	0	0
1	1	1	1	0	1	1

The above-mentioned boolean operations can be defined via their truth tables (see Table 1.1), where the *truth values* (also called *boolean constants*) *TRUE* and *FALSE* are represented by 1 and 0. Every boolean formula φ with n variables represents a boolean function $f_\varphi : \{0,1\}^n \to \{0,1\}$. There

are exactly two boolean functions of arity zero (namely, the two boolean constants), four 1-ary boolean functions (e.g., the identity and the negation), and 16 2-ary boolean functions (e.g., the functions corresponding to the boolean operations $\wedge$, $\vee$, $\Longrightarrow$, and $\Longleftrightarrow$ in Table 1.1). In general, there exist exactly 2^{2^n} n-ary boolean functions, since each of the 2^n possible truth assignments fixes two of them, namely the one with value 0 and the one with value 1 for the given assignment.

The truth value of a boolean formula (i.e., the value of the corresponding boolean function) can be determined from the truth values assigned to its variables. For example, the boolean formula

$$\varphi(x,y,z) = (x \wedge \neg y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z) \tag{1.1}$$

depends on three boolean variables, x, y, and z, and for the truth assignment $(1,1,1)$ to (x,y,z) it can be evaluated as follows:

$$\varphi(1,1,1) = (1 \wedge \neg 1 \wedge 1) \vee (\neg 1 \wedge \neg 1 \wedge \neg 1) = (1 \wedge 0 \wedge 1) \vee (0 \wedge 0 \wedge 0) = 0 \vee 0 = 0,$$

that is, φ is false under this assignment. Using the assignment $(1,0,1)$, however, we obtain

$$\varphi(1,0,1) = (1 \wedge \neg 0 \wedge 1) \vee (\neg 1 \wedge \neg 0 \wedge \neg 1) = (1 \wedge 1 \wedge 1) \vee (0 \wedge 1 \wedge 0) = 1 \vee 0 = 1,$$

so φ is true under this assignment. A boolean formula φ is *satisfiable* if there exists an assignment to its variables such that φ evalutes to true. The formula φ from (1.1) is satisfiable, since it has two satisfying assignments, $(0,0,0)$ and the already mentioned assignment $(1,0,1)$. None of the other six assignments satisfies φ.

A *literal* is a variable, such as x, or its negation, $\neg x$. An *implicant* is a conjunction of literals, such as $(x \wedge \neg y \wedge z)$, and a *clause* is a disjunction of literals, such as $(x \vee y \vee \neg z)$. Formulas like the one in (1.1) are in *disjunctive normal form* (DNF), i.e., they are disjunctions of implicants. Similarly, a formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses.

Every boolean formula can be transformed into an equivalent one in DNF, and also into an equivalent formula in CNF, where the new formulas can be exponentially larger than the given formula, though. Two formulas are said to be *equivalent* if they have the same truth value for all truth assignments to their variables. For example, one can easily check that

$$\begin{aligned}\varphi'(x,y,z) = {} & (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge \\ & (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)\end{aligned} \tag{1.2}$$

is a formula in CNF equivalent to φ from (1.1). If the number of literals in all clauses of a formula in CNF is bounded by a constant k, we say that the formula is in *k-CNF*. For example, the formula φ' from (1.2) is in 3-CNF.

The restriction of the above-defined problem SAT to formulas in k-CNF is denoted by k-SAT.

1.5.2.2 Upper Bounds for SAT

How hard is SAT? Since formulas with n variables have 2^n truth assignments, the naive deterministic algorithm for SAT runs in time $\mathcal{O}(n^2 \cdot 2^n)$: On input φ (a given boolean formula with n variables), this algorithm tests for all possible truth assignments to the variables of φ, one after the other, whether they satisfy φ, and each single test can be done in quadratic time. If a satisfying assignment is found, the algorithm halts and accepts; but it can reject its input only after having tested all 2^n assignments unsuccessfully. This naive algorithm for SAT can be improved, as we will see below.

It is also known that certain restrictions of the problem SAT can be solved more efficiently. For example, if the given formula is in DNF, then it can be decided in polynomial time whether or not it is satisfiable. This is because in this case we can check the satisfiability of the implicants of the formula, one by one; as soon as a satisfiable one is found, the formula is accepted; otherwise, we can safely reject it as soon as the last unsatisfiable implicant has been checked.

If the given formula is in CNF, however, the problem seems to be much harder, namely as hard as the unrestricted problem SAT. Yet even in this case, an efficient algorithm is possible, provided there is no clause with more than two literals in the given formula: Jones, Lien, and Laaser [354] showed that 2-SAT is solvable in polynomial time, and that it belongs to a complexity class that (presumably) is yet smaller than P, namely, to the class of problems solvable by a nondeterministic Turing machine in logarithmic space (see, e.g., [508] for a detailed proof). This class is widely believed to be a proper subclass of P, just as P is widely believed to be a proper subclass of NP.

The runtime of the naive algorithm for SAT was given as a function of the number of variables of the given formula. Why? It would seem more natural to define the size of a formula as the number of *occurrences* of (positive or negated) variables (neglecting the encoding details with respect to parentheses, $\wedge$, $\vee$, and $\neg$ symbols, etc.), and each variable can occur very often in a formula. For example, x, y, and z each occur twice in the formula φ from (1.1), but they each occur six times in the equivalent formula φ' from (1.2). In general, every variable might occur exponentially often (in the number of variables). However, this is only due to the fact that the way we represented these formulas in (1.1) and (1.2) was more wasteful than needed. If one represents a boolean formula by a *boolean circuit* (as in Figure 1.5), then we have a compact representation of the given SAT instance whose size (under a reasonable encoding) is polynomial in the number of variables, and a polynomial blow-up is something we can easily cope with.

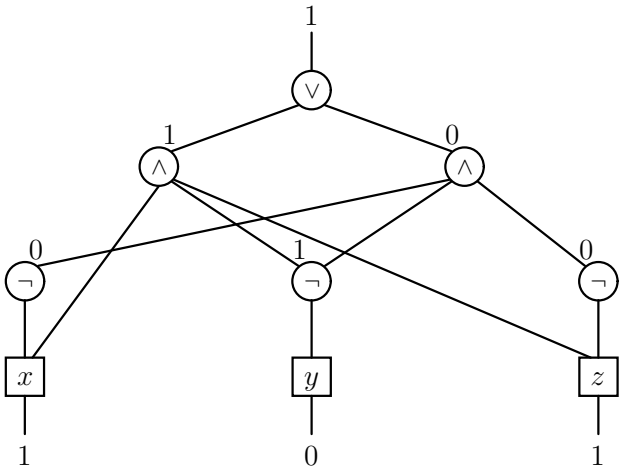


Fig. 1.5 A boolean circuit for the formula in (1.1) with satisfying assignment (1,0,1)

Alternatively, the runtimes of SAT and k-SAT algorithms are also given in the number of clauses or in relation to both parameters, the number of variables and the number of clauses.

As mentioned above, the naive SAT algorithm can be crucially improved. This fascinating, exceedingly important problem has been investigated in computer science for decades, in order to develop better and better SAT solvers. Not much different than in high-performance sports, new records are established for SAT again and again (see, e.g., the surveys by Schöning [527], Woeginger [599], and Riege und Rothe [501] for results on moderately exponential-time algorithm for SAT and other hard problems).

Table 1.2 Some deterministic upper bounds for k-SAT and SAT

Problem	Upper bound	Source
2-SAT	P	Jones, Lien, and Laaser [354]
3-SAT	1.4726^n	Brueggemann and Kern [134]
k-SAT, $k \geq 4$	$\left(2 - \frac{2}{(k+1)}\right)^n$	Dantsin at al. [183]
SAT	$2^{n\left(1-\frac{1}{\log(2m)}\right)}$	Dantsin and Wolpert [184]

Table 1.2 lists some of these upper bounds for k-SAT and SAT, where n is the number of variables and m the number of clauses of the input formula and where polynomial factors are ignored.[7] Table 1.3 shows how the results for 3-SAT have evolved over time. All these upper bounds refer to *deterministic*

[7] It is common to neglect polynomial factors when analyzing exponential runtimes.

algorithms (as opposed to *randomized* algorithms that may be more efficient, but can make errors) and to the *worst case*, i.e., when analyzing the runtimes of these algorithms, one assumes the "worst cases," the most difficult problem instances the algorithm must be able to handle.

Table 1.3 Improving the deterministic upper bounds for 3-SAT

Upper bound	Source
2^n	naive 3-SAT algorithm
1.6181^n	Monien and Speckenmeyer [428]
1.4970^n	Schiermeyer [524]
1.4963^n	Kullmann [378]
1.4802^n	Dantsin et al. [183]
1.4726^n	Brueggemann and Kern [134]

As one can see, all upper bounds listed in Tables 1.2 and 1.3 for these variants of the satisfiability problem are still exponential in the number of variables. Now, what is the point in improving an exponential-time algorithm running in time, say, 2^n to another one with a more moderate exponential runtime of c^n, where c is a constant with $1 < c < 2$? In practical applications, such an improved, more moderate exponential runtime can have a great impact, as one is then able to process significantly larger inputs in the same time. Note that the exponential growth hits only from a certain input size on; therefore, even exponential-time algorithms can be practicable for moderate input sizes below this threshold. Suppose, for instance, that within one hour we can solve all inputs of size up to 30 by an algorithm running in time 2^n. If we are able to improve this algorithm so that our new algorithm runs in time, say, $\sqrt{2}^n \approx 1.4142^n$, then we will now be able to handle inputs up to size 60 on the same computer within one hour, since we have $\sqrt{2}^{60} = 2^{(1/2)\cdot 60} = 2^{30}$. In practice, this really can make a difference.

1.5.2.3 How to Prove Lower Bounds: Reducibility and Hardness

However, how can one prove a *lower* bound for SAT? Indeed, for (unrestricted) deterministic algorithms, no better lower bound than linear time is known for SAT to date (see, e.g., the work of Fortnow et al. [289]). But this lower bound is trivial, as linear time is needed just to read the input. Due to this difficulty to prove lower bounds of problems in the sense of the Ω or ω notation, one takes another approach: One compares the complexity of a given problem with that of other problems in a complexity class and tries to show that it is at least as hard to solve the given problem as it is to solve any of the other problems in the class. If one succeeds, the considered problem is *hard* for the entire complexity class, and also in this sense it is common to

speak of a problem's *lower bound*: The related complexity class provides a lower bound for the problem at hand, since solving any problem in the class is no harder than solving this problem. If the latter belongs to this class in addition, it is said to be *complete* for it.

To compare two given problems in terms of their complexity, we now introduce the notion of reducibility on which the above notions of hardness and completeness are based. Intuitively, a reduction of a decision problem A to a decision problem B means that all instances of A can efficiently be transformed into instances of B such that the original instances are yes-instances of A if and only if their transformations are yes-instances of B. This notion of reducibility is just one among many, but perhaps the most important one, and it is called *polynomial-time many-one reducibility* because different instances of A can be mapped to one and the same instance of B by the transformation (see, e.g., the textbooks by Papadimitriou [462] and Rothe [508] for more details and for other reducibilities some of which will be presented in later chapters).

Definition 1.2 (reducibility, hardness, completeness, and closure). An *alphabet* is a finite, nonempty set of characters (or symbols). Σ^* denotes the set of all strings over the alphabet Σ. A total function $f : \Sigma^* \to \Sigma^*$ is said to be *polynomial-time computable* if there is an algorithm that, given any string $x \in \Sigma^*$, computes the function value $f(x)$ in polynomial time. Let FP denote the class of all polynomial-time computable functions.

Let A and B be two given decision problems (for simplicity, they are assumed to be encoded over the same alphabet Σ, i.e., $A, B \subseteq \Sigma^*$). Let $\mathcal{C}$ be any complexity class.

1. We say that A *is (polynomial-time many-one) reducible to* B (denoted by $A \leq_{\mathrm{m}}^{\mathrm{p}} B$) if there is a function $f \in \mathrm{FP}$ such that for each $x \in \Sigma^*$, $x \in A \iff f(x) \in B$.
2. B is $\leq_{\mathrm{m}}^{\mathrm{p}}$*-hard for* $\mathcal{C}$ (or, shorter, $\mathcal{C}$*-hard*) if $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ for each set $A \in \mathcal{C}$.
3. B is $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete in* $\mathcal{C}$ (or, shorter, $\mathcal{C}$*-complete*) if B is $\mathcal{C}$-hard and in $\mathcal{C}$.
4. $\mathcal{C}$ is *closed under the* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-reducibility* (or, shorter, $\leq_{\mathrm{m}}^{\mathrm{p}}$*-closed*) if for any two problems A and B, it follows from $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and $B \in \mathcal{C}$ that A is in $\mathcal{C}$.

Cook [174] proved that SAT is NP-complete by encoding the computation of an arbitrary NP algorithm on any given input into a boolean formula that is satisfiable if and only if the algorithm accepts its input. This $\leq_{\mathrm{m}}^{\mathrm{p}}$-reduction of an arbitrary NP problem to SAT shows NP-hardness of, and thus a lower bound for, SAT. That SAT belongs to NP—which provides the corresponding upper bound for this problem—is easy to see: Given a boolean formula φ with n variables, an NP algorithm for SAT nondeterministically guesses a truth assignment for φ and then tests deterministically whether the guessed assignment satisfies the given formula. Here, the power of nondeterminism is exploited. In nondeterministic polynomial time, one can guess (and then

deterministically verify) each of the 2^n possible assignments, where the corresponding computation tree has exactly 2^n paths whose length is polynomial in n. If there exists at least one accepting path in this tree (i.e., at least one satisfying assignment), the formula is accepted; otherwise, it is rejected.

SAT is the first natural problem whose NP-completeness could be proven; therefore, Cook's result [174] is a milestone in complexity theory. The notion of NP-comleteness has an immediate relation to the above-mentioned "P = NP?" problem, as the following lemma shows. In addition, we list a number of other fundamental properties of the $\leq_{\mathrm{m}}^{\mathrm{p}}$-reducibility and of the complexity classes P and NP that will be useful throughout the book.

Lemma 1.1. *1.* P *and* NP *are* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-closed.*
2. If $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ *and* B *is in* P*, then* A *is in* P*.*
3. If $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ *and* A *is* NP*-hard, then* B *is* NP*-hard.*
4. P = NP *if and only if* SAT *is in* P*.*

Like most complexity classes, P and NP are closed under $\leq_{\mathrm{m}}^{\mathrm{p}}$-reductions according to the first statement in Lemma 1.1. This is a very useful property that can be applied in many proofs about these classes, such as in the proof of the fourth statement of this lemma, which says that the famous "P = NP?" problem can be solved simply by finding a (deterministic) polynomial-time algorithm for SAT. In this sense, SAT—just as every other NP-complete problem—represents the whole class NP. That is, one could take *any* other NP-complete problem to replace SAT in this equivalence. To see why this equivalence holds, just note that (a) if P = NP then the NP problem SAT is immediately in P, and (b) if SAT—or any other NP-complete problem—were in P, then all the rest of NP would trail behind due to the $\leq_{\mathrm{m}}^{\mathrm{p}}$-closure of P, which immediately would imply equality of P and NP. Similarly simple proofs can be given for the remaining statements of this lemma that are immediate consequences of the definitions, too.

By the second statement of Lemma 1.1, upper bounds are inherited downward in terms of $\leq_{\mathrm{m}}^{\mathrm{p}}$, and by its third statement, lower (NP-hardness) bounds are inherited upward in terms of $\leq_{\mathrm{m}}^{\mathrm{p}}$. That is why the $\leq_{\mathrm{m}}^{\mathrm{p}}$-reducibility is a very useful tool:

- on the one hand, to prove new upper bounds (in particular, new P algorithms), and
- on the other hand, to prove new lower bounds (in particular, new NP-hardness results).

As regards the latter task, the problem SAT is particularly useful to show other problems NP-hard: Starting from a SAT instance, reductions to many other problems can easily be given. And even more suitable are certain restrictions of this problem. For example, the Cook reduction to SAT even provides a boolean formula in CNF. Thus, also the restriction of the satisfiability problem to formulas in CNF is NP-complete. Moreover, if one wants to start from that problem to show other problems NP-hard, it is sometimes

very useful if the given boolean formula is not only in CNF, but is even a 3-CNF, i.e., if every clause of the formula has at most three literals. To this end, we first need to show that this restriction of SAT is NP-hard. That is, we need to give a $\leq_{\mathrm{m}}^{\mathrm{p}}$-reduction from, say, SAT restricted to CNF formulas to SAT restricted to 3-CNF formulas. And that is what we will do now to give a specific example of a $\leq_{\mathrm{m}}^{\mathrm{p}}$-reduction (see also, e.g., [299, 508]).

For any given formula φ in CNF, we want to construct an equivalent formula ψ in 3-CNF. To this end, it is enough to replace every clause φ with more than three literals, say $C = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k)$ with $k \geq 4$ literals $\ell_1, \ell_2, \ldots, \ell_k$, by a new subformula C' that, first, is in 3-CNF and, second, is satisfiable if and only if C is satisfiable. This new subformula will have $k-3$ new variables, $y_1, y_2, \ldots, y_{k-3}$, and will consist of $k-2$ clauses of the following kind:

$$C' = (\ell_1 \vee \ell_2 \vee y_1) \wedge (\neg y_1 \vee \ell_3 \vee y_2) \wedge \cdots \wedge (\neg y_{k-4} \vee \ell_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k).$$

It is easy to see that if C is satisfiable, so is C', since a satisfying assignment for C can by extended to a satisfying assignment for C' as follows:

- If the clause $(\ell_1 \vee \ell_2 \vee y_1)$ is true under the satisfying assignment for C because ℓ_1 or ℓ_2 are satisfied already, then extend this assignment by setting all variables y_i to 0. Obviously, this ensures that every clause of C' is satisfied as well.
- If the clause $(\neg y_{k-3} \vee \ell_{k-1} \vee \ell_k)$ is true under the satisfying assignment for C because ℓ_{k-1} or ℓ_k are satisfied already, then extend this assignment by setting all variables y_i to 1. Obviously, this ensures that every clause of C' is satisfied as well.
- Otherwise, there must exist another clause of C' (neither the first nor the last one) that is true under the satisfying assignment for C, since we know that at least one of the literals ℓ_j is satisfied. Let $(\neg y_{j-2} \vee \ell_j \vee y_{j-1})$ be the first such clause of C'. Now, extending the assignment for C by setting to 1 all variables y_i, $1 \leq i \leq j-2$, and by setting to 0 all the remaining variables, y_i with $j-1 \leq i \leq k-3$, obviously satisfies every clause of C'.

On the other hand, restricting any satisfying assignment for C' to the variables occurring in C (which correspond to the literals $\ell_1, \ell_2, \ldots, \ell_k$) provides a satisfying assignment for C. Therefore, if C' is satisfiable, so is C. It follows that the clause C is equivalent to the subformula C'. Since we introduce for each such clause with more than three literals a new (disjoint) set of new variables, the original formula φ is equivalent to the new formula ψ constructed in this way. It is also clear that the construction can be done in polynomial time. Hence, by the third statement of Lemma 1.1, even 3-SAT is NP-hard.

By the way, sometimes it is useful to start in a reduction from a formula that not only is in 3-CNF, but that even has the property that each clause has *exactly* three literals. Do you see how one would have to modify the

above reduction in order to show that even that restriction of the satisfiability problem is NP-complete?

Garey and Johnson [299] collected hundreds of NP-complete problems from a variety of scientific fields already more than three decades ago; meanwhile, there are probably thousands, if not tens of thousands, problems known to be NP-complete (see also Johnson's related column in the *Journal of Algorithms* [352]). For example, INDEPENDENT-SET is the problem to decide, given a graph G and a positive integer m, whether or not G has an independent set of size at least m.[8] It is easy to see that this problem is in NP. On the other hand INDEPENDENT-SET can be seen to be NP-hard (and, therefore, NP-complete [299]) by a reduction from 3-SAT that, informally stated, works as follows. Given a boolean formula in 3-CNF with m clauses (each of which has exactly three literals), create a triangle for each clause (i.e., create three vertices—each being labeled with one of the literals of that clause—and connect any two of them by an edge), and create an edge between any two vertices of two distinct triangles whenever one is labeled with a literal and the other one with its negation. It is easy to see that this graph has an independent set of size m if and only if the given formula is satisfiable.

We will come across the satisfiability problem, SAT, and its variants in several places later in this book. On the one hand, we will apply its NP-completeness to determine the complexity of some of those problems we are interested in here. On the other hand, in Chapter 6 we will be concerned with boolean formulas in the context of judgment aggregation.

1.5.2.4 Some Background on Approximation Theory

If a decision problem is NP-hard, one cannot expect to find a deterministic polynomial-time algorithm solving it exactly. However, one can try to find efficient approximation algorithms for the optimization variants of this problem, i.e., polynomial-time algorithms that may not find an exact solution to the problem but can approximate an optimal solution within a certain factor. For example, an optimization variant of SAT is the problem MAX-SAT: Given a boolean formula in CNF, find an assignment of its variables that satisfies most of its clauses. An optimization variant of the problem INDEPENDENT-SET defined above is MAX-INDEPENDENT-SET: Given a graph G, output the size of a maximum independent set of G.

Fix some α, $0 < \alpha < 1$. An *α-approximation algorithm* A for a maximization problem is a polynomial-time algorithm such that for each instance x, A outputs a solution to the problem whose value is at least α times the optimal value for x. The value α is called the *approximation factor* (a.k.a. the *approximation ratio* or *performance guarantee*) of an α-approximation algo-

[8] An *independent set* of a graph is a subset I of its vertex set such that no two vertices in I are connected by an edge. The size of an independent set is the number of its vertices.

rithm. Note that α may also depend on the size of the given instance. For minimization problems, the analogous notions are defined for $\alpha > 1$.

As an example, consider the restriction of INDEPENDENT-SET where the degree of each vertex (i.e., the number of incident edges) is bounded by k; denote this problem by k-DEGREE-INDEPENDENT-SET. Note that for $k \geq 4$, this problem is also NP-complete. However, there is a simple $\frac{k}{k+1}$-approximation algorithm for MAX-k-DEGREE-INDEPENDENT-SET: Start with I set to the empty set (which is trivially independent); while there are still vertices in the given graph G, delete an arbitrary vertex v and all its neighbors from G and add v to I. The resulting set I will obviously be independent. Since in each while-loop, some vertex is added to I and at most $k+1$ vertices are deleted from G (due to the bound on the vertex degree), I has size at least the number of G's vertices divided by $k+1$, which is at least $1/(k+1)$ times the size of a maximum independent set of G.

For the general MAX-INDEPENDENT-SET problem (without any given bound on the vertex degree of the given graph), a quite similar algorithm, the so-called *minimum-degree greedy heuristic*, provides a good approximation in certain cases: Given a graph G, pick any vertex of minimum degree, delete it and all its neighbors from G, and repeat this procedure until you are left with an empty graph. Bodlaender, Thilikos, and Yamazaki [79] show that for certain well-behaved classes of graphs (including trees, so-called split graphs, and complete k-partite graphs, for any k), this simple heuristic indeed finds the optimum, i.e., a maximum independent set.

We say an optimization problem has a *polynomial-time approximation scheme* (*PTAS*) if for each ε, $0 < \varepsilon < 1$, there is an α-approximation algorithm for the problem, where $\alpha = 1 - \varepsilon$ if it is a maximization problem, and $\alpha = 1 + \varepsilon$ if it is a minimization problem. A *fully polynomial-time approximation scheme* (*FPTAS*) is a PTAS whose running time is bounded by a polynomial of the input size and of $1/\varepsilon$. For example, it is known that MAX-3-DEGREE-INDEPENDENT-SET has no PTAS [65].

For more background on approximation theory, including techniques to show approximation results as well as techniques to prove inapproximability results (under certain complexity-theoretic hypotheses), we refer to the textbook by Vazirani [583] and the survey by Arora and Lund [13]. Approximation results will be mentioned in most of the following chapters; see, in particular, Sections 8.5.2.2 and 8.5.2.3 starting on page 523.

We conclude the current chapter by giving a brief compendium of complexity classes and we will illustrate these classes by exhibiting suitable variants of SAT that are complete for them.

1.5.3 A Brief Compendium of Complexity Classes

We got to know the complexity classes P and NP already, see Section 1.5.1.2 starting on page 20. As we said there, these are the most central complexity classes with respect to the time complexity measure, and indeed, most of the problems we will come across in this book can be classified to either belong to P or be NP-complete. Moreover, we have already seen that SAT and 3-SAT, too, are NP-complete and that 2-SAT is in P.[9] However, as we will also see completeness results for classes other than NP later in this book, we now briefly describe these classes, along with "their" variants of SAT, and point the reader to the relevant chapter or section for more details.

1.5.3.1 Polynomial Space

SAT is the satisfiability problem for propositional formulas with no quantifiers involved. Its quantified variant is the following:

QUANTIFIED-BOOLEAN-FORMULA (QBF)	
Given:	$H = (\exists X_1)(\forall X_2)\cdots(\mathfrak{Q}X_k)[\varphi(X_1, X_2, \ldots, X_k)]$, a quantified boolean formula, where the X_i are disjoint sets of variables, φ is a boolean formula without quantifiers, all variables occurring in φ are quantified, and $\mathfrak{Q} = \exists$ if k is odd, and $\mathfrak{Q} = \forall$ if k is even.
Question:	Does H evaluate to true, i.e., does there exist a truth assignment to the variables in X_1 such that for each truth assignment to the variables in X_2, there exists ... a truth assignment to the variables in X_k such that φ is true under this assignment?

Because of its quantifier structure QBF seems to be much harder a problem than SAT; in fact, it is PSPACE-complete, where PSPACE is the class of problems that can be solved by a Turing machine[10] in polynomial space; alternatively, PSPACE is the class of problems that can be solved by an *alternating Turing machine* in polynomial *time* (see the work of Chandra, Kozen, and Stockmeyer [148]).

We will come across PSPACE-complete problems in Section 2.3.1.4 on page 88, where we consider the hardness of detecting whether a player in a combinatorial two-player game with perfect information has a winning strategy. We will also see PSPACE-complete problems in the context of online manipulation in sequential elections (see Section 4.3.3.10 starting on page 284).

[9] We won't consider completeness for P here. Note that $\leq_m^p$-completeness in P is trivial, as $\leq_m^p$ is too coarse for P and smaller classes (see, e.g., [508, Lemma 3.37], which says that *every* P set distinct from $\emptyset$ and Σ^* is $\leq_m^p$-complete in P). There are also nontrivial completeness results for this class, but with respect to other reducibilities than $\leq_m^p$.

[10] It does not matter whether this Turing machine is deterministic or nondeterministic; a consequence of Savitch's theorem [522] is that nondeterministic polynomial space coincides with deterministic polynomial space.

1.5.3.2 The Polynomial Hierarchy

Restricting QBF to a fixed number of i alternating quantifiers gives the problems QBF_i (starting with an existential quantifier) and their complements, $\overline{\mathrm{QBF}_i}$ (starting with a universal quantifier). Note that QBF_1 is nothing other than SAT. The corresponding hierarchy of complexity classes has been introduced and studied by Meyer and Stockmeyer [424, 562]: the so-called *polynomial hierarchy*, $\mathrm{PH} = \bigcup_{i \geq 0} \Sigma_i^p$. Its Σ_i^p levels are inductively defined by

$$\Sigma_0^p = \mathrm{P}, \quad \Sigma_1^p = \mathrm{NP}, \quad \text{and} \quad \Sigma_i^p = \mathrm{NP}^{\Sigma_{i-1}^p},\ i \geq 2,$$

where "$\mathrm{NP}^{\Sigma_{i-1}^p}$" means that an NP "oracle Turing machine" accesses an "oracle" from Σ_{i-1}^p (see page 255 and [508, Def. 2.22] for more details). Since these classes are not likely to be closed under complementation, one has also studied their co-classes, the Π_i^p levels of the polynomial hierarchy, defined by $\Pi_i^p = \mathrm{co}\Sigma_i^p = \{\overline{L} \,|\, L \in \Sigma_i^p\}$, $i \geq 0$.

In particular, $\Pi_0^p = \mathrm{P}$ (because, as a deterministic class, P clearly is closed under complementation) and $\Pi_1^p = \mathrm{coNP}$ is the class of complements of NP problems. Just as with the "$\mathrm{P} = \mathrm{NP}$?" question, it is open whether or not NP equals coNP, but it is widely believed that these two classes are distinct. It is also widely believed (but as yet unproven) that the polynomial hierarchy is strict, i.e., that all their levels differ from each other. Note that $\mathrm{P} = \mathrm{NP}$ would imply $\mathrm{NP} = \mathrm{coNP}$, and even that $\mathrm{PH} = \mathrm{P}$. More generally, a collapse of any level of this hierarchy ($\Sigma_i^p = \Pi_i^p$, $i \geq 1$, or $\Sigma_i^p = \Sigma_{i+1}^p$, $i \geq 0$) would imply an upward collapse of the entire hierarchy to this level ($\mathrm{PH} = \Sigma_i^p$).

It is known that for each $i > 0$, QBF_i is Σ_i^p-complete and $\overline{\mathrm{QBF}_i}$ is Π_i^p-complete. It is also known that the levels of the polyomial hierarchy can be characterized via alternating, polynomially length-bounded existential and universal quantifiers (where the Σ_i^p levels start with an existential and the Π_i^p levels with a universal quantifier), which nicely mirrors the structure of QBF_i and $\overline{\mathrm{QBF}_i}$ formulas, respectively; see the original work of Meyer and Stockmeyer [424, 562] or the textbook by Rothe [508, Section. 5.2] for a proof of this characterization.

Completeness for the levels of the polynomial hierarchy, specifically for Σ_{2k}^{p}, $k \geq 1$, will play a role in Section 3.3.1.3 on page 174 when we will be concerned with bribery in path disruption games; in Section 3.3.3 starting on page 183 when we will study stability problems in hedonic games; in, again, Section 4.3.3.10 starting on page 284 when we will consider online manipulation in sequential elections; in Section 6.3 starting on page 374 when we will study the complexity of problems related to judgment aggregation; and in Section 8.5.2.5 starting on page 529 when we turn to the complexity of problems related to Pareto efficiency (see Definition 8.2 on page 509) and envy-freeness (see Definition 8.3 on page 510) in the allocation of indivisible goods.

The Δ_i^p levels of the polynomial hierarchy are $\Delta_0^p = \mathrm{P}$ and $\Delta_i^p = \mathrm{P}^{\Sigma_{i-1}^p}$ for $i \geq 1$; note that $\Delta_1^p = \mathrm{P}^{\mathrm{P}} = \mathrm{P} = \Delta_0^p$. Once again related to online manipulation in sequential elections, we will specifically come across the class $\mathrm{P}^{\mathrm{NP}} = \Delta_2^p$ in Section 4.3.3.10 starting on page 284. In addition, we will also see completeness for the restriction $\mathrm{P}^{\mathrm{NP}[1]}$ of P^{NP} with only one oracle query allowed. Relatedly, the restriction of P^{NP} to logarithmically many oracle queries, denoted by $\mathrm{P}^{\mathrm{NP}[\log]}$, is known to be equivalent (due to results of Hemachandra [320] and Köbler, Schöning, and Wagner [370]) to *"parallel access to* NP*,"* denoted by $\mathrm{P}_{\|}^{\mathrm{NP}}$ and formally defined on page 255. $\mathrm{P}_{\|}^{\mathrm{NP}}$-completeness will be relevant to the winner problems for Dodgson, Young, and Kemeny elections in Section 4.3.1.3 starting on page 255. Another problem known to be complete for this class is the problem of whether the minimum-degree greedy heuristic (described in Section 1.5.2.4 starting on page 31) can approximate the size of a maximum independent set of a given graph within a constant factor [332] (see the work of Hemaspaandra, Rothe, and Spakowski [333] for the analogous results regarding the problem Min-Vertex-Cover[11] and the so-called *edge deletion heuristic* and the *maximum-degree greedy heuristic*).

The following problem is a variant of the satisfiability problem that (by a result of Wagner [585]; see also [331] for some technical tweaks in the proof) is known to be complete in P^{NP}:

Max-Satisfying-Assignment-Equality (Max-SAT-ASG$_=$)	
Given:	Two boolean formulas in CNF with exactly three literals per clause.
Question:	Do they have the same maximal satisfying assignment?

And an example of a variant of the satisfiability problem that (again due to Wagner [585]) is known to be $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete is the problem Odd-SAT: Given a list of boolean formulas, is it true that the number of satisfiable formulas in the list is odd?

1.5.3.3 DP: the Second Level of the Boolean Hierarchy over NP

There are also other hierarchies built upon NP, for example, the *boolean hierarchy over* NP (see, e.g. [136, 137, 508, 500]). In this book, we will only be interested in the second level of this hierarchy, the class DP, which was introduced by Papadimitriou and Yannakakis [464] and is defined as the class of differences of any two NP sets. An example of a variant of the satisfiability problem that is DP-complete is the problem SAT-UNSAT, which contains all pairs (φ, ψ) of boolean formulas such that φ is satisfiable and ψ is not.

We will come across further DP-complete problems that are related to the exact variant of the margin of victory problem in voting (see Section 4.3.5.4

[11] Min-Vertex-Cover is defined as the problem: Given a graph G, output the size of a minimum vertex cover of G, where a *vertex cover of* G is a subset of the vertices of G that contains at least one of the two vertices of each edge of G.

starting on page 321) and to exactly maximizing social welfare in allocating indivisible goods (see Sections 8.5.2.2 and 8.5.2.3 starting on page 523).

1.5.3.4 Probabilistic Polynomial Time

MAJORITY-SAT is the problem of whether a given boolean formula with n variables has at least 2^{n-1} satisfying assignments, i.e., at least half of the total number of assignments are required to be satisfying assignments. The complexity class that naturally corresponds to this problem has been introduced by Gill [305] as *probabilistic polynomial time*, abbreviated by PP. He defines PP via *probabilistic (polynomial-time) Turing machines*, which can be viewed as nondeterministic (polynomial-time) Turing machines that accept their input if and only if at least half of their total number of computation paths accept. In Section 3.3.2.2 on page 178, we will see that certain problems related to the complexity of "false-name manipulation" in weighted voting games (defined in Section 3.2.3) are PP-complete.

Overview

The landscape of the above-defined complexity classes is shown in Figure 1.6 as a Hasse diagram, i.e., every ascending line from some complexity class $\mathcal{C}$ to some other class $\mathcal{D}$ indicates an inclusion: $\mathcal{C} \subseteq \mathcal{D}$. Most of these inclusions follow immediately from the definitions; some require a nontrivial proof. For example, the inclusion $\mathrm{P}_{\|}^{\mathrm{NP}} \subseteq \mathrm{PP}$ follows from the closure of PP under so-called parity reductions that was shown by Beigel, Hemachandra, and Wechsung [64]. Presumably, P^{NP} is not contained in PP, and PP and PH are incomparable. None of the inclusions depicted in Figure 1.6 is known to be strict, though it is widely suspected that they are.

In later chapters, some further complexity classes will be introduced and discussed (e.g., those depicted in Figure 2.25 on page 129 or the parameterized complexity class W[2] occurring in Table 6.5 on page 387). However, as will be explained there, these classes are of a different type (those from Figure 2.25 are classes of functions and not classes of sets, while parameterized complexity classes contain *parameterized* problems that, unlike regular decision problems, specify one or more parameters in addition to the input) and, therefore, cannot directly be compared with those in Figure 1.6.

1.5.3.5 And Now, Finally, ...

... let's get ready to play, vote, and divide!

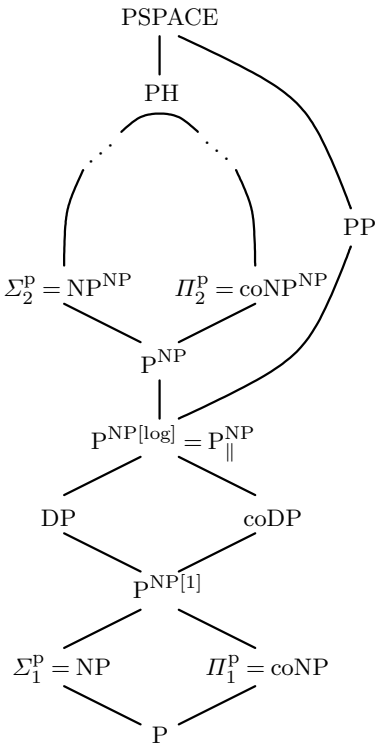


Fig. 1.6 Inclusions between some complexity classes