Progress in Computer Science and Applied Logic
26

Manindra Agrawal
Vikraman Arvind
Editors

Perspectives in Computational Complexity

The Somenath Biswas Anniversary Volume

Birkhäuser

Progress in Computer Science and Applied Logic

Volume 26

Editor-in-Chief

Erich Grädel, Aachen, Germany

Associate Editors

Eric Allender, Piscataway, NJ, USA
Mikołaj Bojańczyk, Warsaw, Poland
Sam Buss, San Diego, CA, USA
John C. Cherniavski, Washington, DC, USA
Javier Esparza, Munich, Germany
Phokion G. Kolaitis, Santa Cruz, CA, USA
Jouko Väänänen, Helsinki, Finland and Amsterdam, The Netherlands

For further volumes:
http://www.springer.com/series/4814

Manindra Agrawal · Vikraman Arvind
Editors

Perspectives in Computational Complexity

The Somenath Biswas Anniversary Volume

Birkhäuser

Editors
Manindra Agrawal
Department of Computer Science
and Engineering
Indian Institute of Technology
Kanpur
India

Vikraman Arvind
CIT Campus
Institute of Mathematical Sciences
Chennai
India

ISBN 978-3-319-05445-2 ISBN 978-3-319-05446-9 (eBook)
DOI 10.1007/978-3-319-05446-9
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014942531

Mathematics Subject Classification (2010): 03-XX, 03D15, 68-XX, 68Q15

© Springer International Publishing Switzerland 2014
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.birkhauser-science.com)

Contributors

Eric Allender Department of Computer Science, Rutgers University, New Brunswick, NJ, USA

Vikraman Arvind Institute of Mathematical Sciences, Chennai, India

S. Ajesh Babu Microsoft Research India, Bangalore, India

Markus Bläser Computer Science, Saarland University, Saarbrücken, Germany

Sumanta Ghosh Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, Uttar Pradesh, India

Neeraj Kayal Microsoft Research, Bangalore, India

Piyush P. Kurur Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, Uttar Pradesh, India

Meena Mahajan The Institute of Mathematical Sciences, Chennai, India

Bruno Poizat Institut Camille Jordan, Université Claude Bernard, Villeurbanne-cedex, France

Jaikumar Radhakrishnan School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai, India

Ramprasad Saptharishi Microsoft Research, Bangalore, India

Nitin Saxena Department of CSE, IIT Kanpur, Kanpur, India

Jacobo Torán Department of Theoretical Computer Science, University of Ulm, Ulm, Germany

N. Variyam Vinodchandran Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

Somenath Biswas 2013
With the permission of © Somenath Biswas

Preface

In the summer of 2012, we organized a three day "Computational Complexity" workshop at the Indian Institute of Technology, Kanpur, India, in honor of Professor Somenath Biswas to celebrate his 60th birthday. It was a fitting event for the occasion, well-attended by several well-known experts in the field from different parts of the world.

Professor Biswas is one of the first complexity theorists from India. In his teaching and research career spanning over 30 years, apart from doing quality research, he has contributed immensely to the development of the field in India. We felt that to bring out a *festschrift* volume of articles in complexity theory, based partly on the talks at the workshop, would be a lasting tribute. We are deeply grateful to the eminent researchers who enthusiastically agreed to contribute articles for this project and also helped, in a cross-refereeing process, with refereeing each other's contributed articles. These articles span different aspects of recent complexity theory research, including the isomorphism conjecture, arithmetic circuit complexity, space-bounded complexity classes, proof complexity, applications of entropy, and the complexity of graph isomorphism. As former students of Somenath Biswas, we feel privileged to edit this volume. It is an expression, as it were, of our affection and regard for him.

We are grateful to Eric Allender for his valuable advice and support from the early stages of this book project, and for suggesting the "Progress in Computer Science and Applied Logic" Springer-Birkhäuser series. We would also like to thank Erich Grädel, the chief editor of the series for enthusiastically supporting the project.

March 2014

Manindra Agrawal
Vikraman Arvind

Contents

Chapter 1
Complexity Theory Basics: NP and NL

Vikraman Arvind

Abstract We introduce basic concepts and results in computational complexity as background for some of the articles in this volume. Our focus is on the complexity classes nondeterministic polynomial time (NP) and nondeterministic logarithmic space (NL). The presentation is aimed at computer science students at a senior undergraduate level, and assumes some familiarity with algorithm design and theory of computation. The material is covered at a fairly brisk pace. Several results and proof details are incorporated in exercises which the reader is urged to solve or look up in textbooks such as [BDG88, Pap94, AB09].

Keywords Nondeterministic polynomial time · NP-completeness · Nondeterministic logarithmic space

Mathematics Subject Classification (2010) Primary 68Q15.

1.1 NP-completeness

The story of modern computational complexity begins with the advent of stored program computers in the 1940s and the need for efficiently solving optimization problems by programming the computer. It was soon discovered that a brute-force enumerative search for the optimal solution yields only algorithms that take exponential time, since the number of candidate solutions for optimization problems is typically exponential in the input size. Such solutions were not practical even for inputs of moderate size. Cobham [Cob64] and Edmonds [Edm65], around 1965, independently suggested polynomial-time boundedness as an appropriate theoretical criterion for efficient computation. This was an important conceptual contribution.

V. Arvind (✉)
Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai 600113, India
e-mail: arvind@imsc.res.in

M. Agrawal and V. Arvind (eds.), *Perspectives in Computational Complexity*,
Progress in Computer Science and Applied Logic 26, DOI: 10.1007/978-3-319-05446-9_1,
© Springer International Publishing Switzerland 2014

Although algorithms with linear or quadratic time bounds are desirable in practice, polynomial time computation is theoretically satisfactory for several reasons. First, it rules out exhaustive search solutions which are typically exponential time. Also, since polynomials are closed under composition, it makes polynomial-time bounded computation closed under procedure calls. Thus, a polynomial-time bounded program making calls to a library of polynomial-time subroutines is still polynomial time. Furthermore, as reasonable models of computation can simulate each other with at most a polynomial-time slowdown,[1] it makes the notion of polynomial-time solvability independent of the computation model.

The next big step was the pioneering research of Cook [Coo71], Levin [Lev73], and Karp [Kar72]. The class NP, consisting of decision problems that have nondeterministic polynomial-time decision procedures, was identified as the class that captures most natural optimization problems of interest. The notion of polynomial-time reductions was used to compare the relative difficulty of problems. Propositional formula satisfiability was shown to be NP-complete under polynomial-time reductions [Coo71, Lev73]. Then several decision problems, arising from optimization problems, were shown NP-complete [Kar72]. NP-complete problems are the hardest problems in the class NP as opposed to decision problems that are polynomial-time solvable.

Let Σ denote a fixed finite alphabet $|\Sigma| \geq 2$. Input instances of decision problems are encoded as finite strings over Σ. Thus, Σ^* comprises of all input instances for a decision problem and the "yes" instances $A \subseteq \Sigma^*$ form a language. Therefore, we can identify decision problems with languages and we use the terms interchangeably. As is customary in computation theory, we will use the standard Turing machine model as the model of computation (see e.g. [HU79]).

Definition 1.1 Let $A, B \subseteq \Sigma^*$ be languages.

1. Then A is said to be polynomial-time *many-one reducible* to B, denoted $A \leq_m^p B$, if there is a polynomial-time computable function f such that for all $x \in \Sigma^*$

$$x \in A \text{ if and only if } f(x) \in B.$$

2. More generally, A is said to be polynomial-time *Turing reducible* to B if there is a polynomial-time bounded oracle Turing machine M such that M^B accepts the language A.

A language $L \subseteq \Sigma^*$ is in the complexity class P if there is a polynomial-time bounded deterministic Turing machine (equivalently, a polynomial-time algorithm) for checking membership in L.

A language $L \subseteq \Sigma^*$ is in the complexity class NP if there is a language $A \in$ P and a polynomial p such that for all $x \in \Sigma^*$

$$x \in L \text{ if and only if } \exists y \in \Sigma^{\leq p(|x|)} \ : \ \langle x, y \rangle \in A,$$

[1] This is a polynomial-time version of the Church-Turing thesis known as the feasibility thesis [vEB90].

where $\Sigma^{\leq p(|x|)}$ denotes strings of length at most $p(|x|)$ over Σ. In other words, the complexity class NP consists of languages L such $x \in L$ has a polynomial-size certificate y of membership in L, and the certificate is polynomial-time verifiable by checking if $\langle x, y \rangle \in A$.

Exercise Show that the following decision problems are in the class NP:

1. Given an undirected graph G and a number k as input, decide if G has a clique of size at least k (known as the CLIQUE problem). The graph G is given by either its adjacency list or adjacency matrix.
2. Given a positive integer m (encoded in binary) decide if it is composite.
3. Given a system of linear equations $Ax = b$ over rationals, where the rational entries of the matrix A and column vector b are encoded in binary, decide if it has a solution.

We now formally define NP-completeness. A language $L \subseteq \Sigma^*$ is said to be NP-complete if L is in NP and each $L' \in$ NP is polynomial-time many-one reducible to L.

Since polynomial-time reducibility between languages is a transitive relation, once a language L' is shown NP-complete, it suffices to show that L' is polynomial-time many-one reducible to L in order to prove that the language L in NP is also NP-complete. A problem that can be directly shown NP-complete is the following:

$$K = \{\langle M, x, 1^t \rangle \mid M \text{ accepts } x \text{ in at most } t \text{ steps}\},$$

where M in the above definition denotes the Turing machine code (as a list of quintuples) of a *nondeterministic* Turing machine.

Exercise Show that K is NP-complete. (Hint: In order to show K is in NP you will need to use a suitable universal Turing machine).

But the first problem shown NP-complete by Cook and Levin was a natural problem, known as the satisfiability problem for propositional formulas, which opened the floodgate to NP-complete problems and the subject of computational complexity.

Theorem 1.2 (Cook-Levin theorem) [Coo71, Lev73] *The satisfiability problem for propositional formulas is NP-complete.*

A propositional formula F is in *conjunctive normal form* (CNF in short) if $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ where each C_i is an OR of variables or their negations. The proof of the Cook-Levin theorem actually shows the stronger result that the satisfiability problem for propositional CNF formulas is NP-complete.

Clearly, NP-complete problems are the hardest problems in NP and all NP-complete problems are polynomial-time equivalent. That is to say, if a polynomial-time algorithm is discovered for any NP-complete problem then we have a polynomial-time algorithm for any problem in NP. Whether P equals NP or not is the central open problem in computational complexity.

Exercise

1. Show that the CLIQUE problem (defined in Exercise 2) isNP-complete by giving a reduction to it from propositional CNF formula satisfiability.
2. A *vertex cover* for an undirected graph G is a subset S of vertices such that for each edge (u, v) of G we have $\{u, v\} \cap S \neq \emptyset$. Given an undirected graph G and a number k as input the VC problem is to decide if G has a vertex cover of size at most k. Show that VC is NP-complete. The graph G is given by either its adjacency list or adjacency matrix.

1.2 Inside NP

Thus, within the class NP we have polynomial-time solvable problems on the one hand, which is the subclass P. At the other extreme, we have NP-complete problems, of which there are abundantly many [GJ79], because most natural optimization problems that arise in practice and we wish to solve efficiently turn out to be NP-complete. A natural question that arises is whether NP contains other problems. The answer to this question is given by Ladner's theorem which states that if P $\neq$ NP then there are problems in NP that are neither in P nor NP-complete. We discuss a proof attributed to Russell Impagliazzo [DF03].

Theorem 2.1 (Ladner's theorem)[Lad75] *If* P $\neq$ NP *there is a problem* $A \in$ NP *that is neither in* P *nor* NP*-complete.*

Proof By assumption the NP-complete problem SAT is not in P. Following standard notation [BDG88, Pap94], let DTIME$[g(n)]$ denote the class of languages accepted by deterministic Turing machines that halt in time bounded by $g(n)$ on inputs of length n. Now, if we knew that SAT $\notin$ DTIME$[g(n)]$ for some fixed superpolynomial function $g(n)$, for example $g(n) = n^{\log n}$, then we can easily find a language $A \in$ NP that is neither in P nor NP-complete. Indeed, let

$$A := \{x01^{|x|^{\log\log|x|}} \mid x \in \text{SAT}\}.$$

Clearly, $A \in$ NP because given a string of the form $x01^k$ we can guess and verify a satisfying assignment for the SAT instance x and check in polynomial time that the pad 1^k is of length $|x|^{\log\log|x|}$. Suppose A is NP-complete. Then SAT $\leq_m^p A$ via some polynomial-time computable reduction f. Notice that

$$x \in \text{SAT} \iff f(x) = x'01^{|x'|^{\log\log|x'|}} \in A \iff x' \in \text{SAT}.$$

Since f is polynomial-time computable, $|f(x)| \leq |x|^c$ for some constant c and hence $|x'| < |x|$ for all but finitely many instances $x \in$ SAT. Thus, it suffices to check if the smaller instance $x' \in$ SAT. Repeatedly applying this argument gives a polynomial-time procedure for SAT contradicting P $\neq$ NP. Hence A cannot be NP-complete.

On the other hand, we claim that $A \notin \mathrm{P}$. For, if A were in P that would give a $|x|^{O(\log\log|x|)}$ time algorithm to decide if $x \in \mathrm{SAT}$, contradicting the assumption that SAT is not in $\mathrm{DTIME}[n^{\log n}]$.

Unfortunately, we can only assume that SAT is not in P and cannot make the stronger assumption that SAT is not in $\mathrm{DTIME}[n^{\log n}]$. So we need to define the padded language A more carefully to make the above argument work. Let

$$A := \{x01^k \mid x \in \mathrm{SAT},\ \ k = f(|x|)\},$$

where the padding function $f(n)$ will be computable in time polynomial in n and constructed by diagonalization. Let $\{M_i\}_{i>0}$ be a recursive enumeration of polynomial-time clocked, deterministic Turing machines; more precisely, let M_i be clocked to run for $n^i + i$ steps on length n inputs. The function f is defined as follows:

1. $i := 1$.
2. For $n := 1$ to ∞ do
3. Let $f(n) = n^i$.
4. If there is an input x of length at most $\log n$ such that $M_i(x)$ accepts and $x \notin A$ or $M_i(x)$ rejects and $x \in A$ then $i := i + 1$.
5. endfor

Notice that at the nth iteration of the for-loop, in which $f(n)$ gets defined, the function $f(m)$ is already defined for $m < n$ and hence checking $x \in A$ for $\log n$ length inputs is well defined. Moreover, checking membership of $x \in A$ can be done in polynomial in n time since $|x| \leq \log n$. Hence, f is defined by the above procedure for all n and is computable in time polynomial in n. This defines the set A.

Suppose $A \in \mathrm{P}$. Then $A = L(M_i)$ for some machine M_i. By construction, there are constants n_0 and $k > i$ such that $f(n) = n^k$ for all $n > n_0$. That means, for all but finitely many input lengths, SAT is polynomial-time reducible to A by the map $x \mapsto x01^{|x|^k}$ which contradicts the assumption $\mathrm{P} \neq \mathrm{NP}$. It follows that for each constant i we have $f(n) > n^i$ for all but finitely many n.

Suppose A is NP-complete and g is a polynomial-time reduction from SAT to A. We will give a polynomial-time algorithm for SAT contradicting the assumption. Since g is polynomial-time computable, we have $|g(x)| \leq |x|^c$ for some constant $c > 0$ and all but finitely many inputs x. If $g(x)$ is not of the form $y01^{f(|y|)}$ we can reject x. Also, if $g(\mathrm{SAT})$ is finite then SAT is trivially in P by table look-up. Suppose $g(\mathrm{SAT})$ is infinite. Then for all but finitely many $x \in \mathrm{SAT}$ we have $g(x) = y01^{f(|y|)}$ where $f(|y|) > |y|^c$. The finitely many exceptions we can keep in a table. Thus, given a SAT instance x we first compute $g(x) = y01^{f(|y|)}$. If x is not in the look-up table, since $f(|y|) < |g(x)| \leq |x|^c$, it follows that $|y| < |x|$ and $x \in \mathrm{SAT}$ if and only if $y \in \mathrm{SAT}$. We can now recurse on the instance y. Overall this gives a polynomial-time SAT algorithm contradicting the assumption. This concludes the proof. □

Exercise Suitably adapt the above proof to show for any language $A \notin \mathrm{P}$ that there is a language $B \notin \mathrm{P}$ such that $B \leq_m^p A$ but $A \not\leq_m^p B$.

1.2.1 The Class NP ∩ coNP

The class coNP consists of languages L such that $\Sigma^* \setminus L$ is in NP.

Indeed, for any class of languages $\mathcal{C}$ we can define co$\mathcal{C}$:

$$\text{co}\mathcal{C} = \{L \subseteq \Sigma^* \mid \Sigma^* \setminus L \in \mathcal{C}\}.$$

Remark 2.2 The class coNP consists of all languages whose complements are in NP. For example $\overline{\text{SAT}}$ is in coNP and, by virtue of SAT being NP-complete, $\overline{\text{SAT}}$ is coNP-complete under polynomial-time many-one reductions. The set TAUT consisting of all propositional tautologies is also coNP-complete (exercise: verify this). Whether NP equals coNP is a major open problem. We can view the NP versus coNP question from the logical perspective of propositional proof systems. For any language $L \in$ NP, by definition for each $x \in L$ there is a polynomial-length proof of membership that can be checked in polynomial time. This can be thought of as a "sound and complete proof system" for L. Thus, the question whether NP = coNP amounts to asking if there is a proof system for propositional tautologies in which all tautologies have polynomial length proofs. This leads to a study of propositional proof systems of different strengths with the aim of proving lower bounds for proof lengths in the proof systems. The article by Jacobo Torán in this volume presents aspects of this fascinating topic with pointers to current research and open problems.

We will now discuss decision problems that are in NP ∩ coNP. For any $L \in$ NP ∩ coNP there are languages A and B in P and a polynomial p such that for each $x \in \Sigma^*$

$$\begin{aligned} x \in L &\leftrightarrow \exists y \in \Sigma^{p(|x|)} \langle x, y\rangle \in A \\ &\leftrightarrow \forall z \in \Sigma^{p(|x|)} \langle x, y\rangle \in B \end{aligned}$$

These are the so-called well-characterized problems. They are well characterized in the sense that membership of x in L can be characterized using an existentially quantified predicate, and can also be characterized using a universally quantified predicate. It is a remarkable phenomenon in complexity theory that the discovery of such characterization often precedes (even anticipates) the discovery of a polynomial-time algorithm for the problem. We discuss a few well-known examples.

The language PM = $\{G \mid G$ has a perfect matching$\}$ has a polynomial-time algorithm as shown in the already mentioned famous paper of Edmonds [Edm65]. However, in the 1940s Tutte, in his well-known theorem stated below, had anticipated this by "well-characterizing" the PM problem.

Theorem 2.3 (Tutte's 1-factor theorem) [Tut47] *An undirected graph G has a perfect matching if and only if for every subset S of the vertex set the number of odd-size components in the graph $G \setminus S$ is bounded by $|S|$.*

Similarly, Farkas' lemma [Sch98, Sect. 7.3] stated below "well-characterizes" linear programming which was shown to be in polynomial time many decades later by Kachiyan [Sch98, Chap. 13].

Theorem 2.4 (Farkas' Lemma) *The system of linear inequalities* $Ax \leq b$ *has a solution if and only if for all* $y^T \geq 0$ *if* $y^T A = 0$ *then* $y^T b \geq 0$.

Exercise Use the characterizations in Theorems 2.3 and 2.4 to show that the perfect matching problem and feasibility of linear inequalities problem are in NP ∩ coNP.

Exercise Show that NP = coNP if and only if some problem in NP ∩ coNP is NP-complete.

The above discussion might lead one to believe that perhaps NP ∩ coNP equals P. However, there are problems in NP ∩ coNP that have defied all attempted polynomial-time solutions. The decision version of the Integer factoring problem is a noteworthy example. Each positive integer n can be uniquely factorized as $n = p_1^{e_1} p_2^{e_2}, \ldots, p_k^{e_k}$ where $p_1 < p_2 < \cdots < p_k$ are distinct primes. Let $enc(n)$ denote an encoding in binary of this factorization. Consider the language

$$\text{FACT} = \{\langle n, i, b\rangle \mid \text{ the } i^{th} \text{ bit of } enc(n) \text{ is } b\}.$$

Exercise

1. Assuming primality testing is in P show that FACT is in NP ∩ coNP.
2. Show that the integer factoring problem can be solved in polynomial time with calls to a decision procedure for FACT.

1.3 The Berman-Hartmanis Conjecture

Polynomial-time reductions define a natural order $\leq_m^p$ on NP languages that raises some fundamental questions about the structure of NP languages.

Exercise Show that the order $\leq_m^p$ on NP languages is a binary relation that is reflexive and transitive but not symmetric.

In order to obtain a partial order from $\leq_m^p$ we define the equivalence relation $A \equiv_m^p B$ if and only if $A \leq_m^p B$ and $B \leq_m^p A$.

Exercise

1. Show that $\equiv_m^p$ is an equivalence relation on NP.
2. For $A \in$ NP let $[A]$ denote the equivalence class containing A for the equivalence relation $\equiv_m^p$. Show that $\leq_m^p$, suitably defined on the equivalences classes $[A]$ for $A \in$ NP, yields a partial order.

This partial order has its top element as [SAT], the equivalence class of all NP-complete languages. Its bottom element is P. Notice that class P contains, among all polynomial-time solvable decision problems, all finite languages. In contrast, assuming P $\neq$ NP, all sets in [SAT], being NP-complete, are infinite.

Definition 3.1 [BH77] Let $A, B \subseteq \Sigma^*$. A polynomial-time many-one reduction f from A to B is a *polynomial-time isomorphism* if $f : \Sigma^* \to \Sigma^*$ is a bijection and f^{-1} is also polynomial-time computable.

Berman and Hartmanis [BH77], in 1977 conjectured that all NP-complete sets are polynomial-time isomorphic to each other. Since they could show [BH77] many natural NP-complete problems to be isomorphic, empirically this appears plausible. Although the conjecture is not currently believed to be true, it gave impetus and direction to a lot of interesting complexity theory research. The article by Eric Allender in this volume surveys the interesting complexity theory research related to the Berman-Hartmanis conjecture over the last two decades. Our aim here is to provide some useful background.

A polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ is *1-invertible* if f is injective and f^{-1} is also polynomial-time computable. More precisely, there is a polynomial-time algorithm that on input $y \in \Sigma^*$ computes $f^{-1}(y)$ if y is in the range of f and outputs $\perp$ otherwise.

Now, suppose A and B are NP-complete languages such that A is reducible to B via a 1-invertible function f and B is reducible to A via a 1-invertible function g, can we then conclude that A and B are polynomial-time isomorphic. The motivation for this approach is its analogy to the setting of the Schröder-Bernstein theorem in set theory which we recall with a quick proof in order to generalize it to the isomorphism setting.

Theorem 3.2 (Schröder-Bernstein theorem) *Let A and B be sets and $f : A \to B$ and $g : B \to A$ be injective functions. Then there is a bijection between A and B.*

Proof The proof idea involves examining the alternating *preimage sequence* of points $x, g^{-1}(x), f^{-}(g^{-1}(x)), \ldots$ for each $x \in A$. Since both f and g are injective, notice that for any $x \in A$ and $y \in B$ the preimages $g^{-1}(x)$ and $f^{-1}(y)$, if they exist, are unique. We can partition A into three parts A_1, A_2, and A_3. The part A_1 consists of $x \in A$ such that the preimage sequence is finite and ends in A. The part A_2 consists of $x \in A$ such that it has a finite preimage sequence ending in B, and A_3 consist of the elements $x \in A$ whose preimage sequence is infinite. Likewise, B is partitioned into three parts B_1, B_2, and B_3 of elements $y \in B$ whose pre-image sequence either ends in A or in B or is infinite, respectively. Define a function $h : A \to B$ as follows:

$$\begin{aligned} \forall\, x \in A_1 \quad & h(x) = f(x), \\ \forall\, x \in A_2 \quad & h(x) = g^{-1}(x), \\ \forall\, x \in A_1 \quad & h(x) = f(x). \end{aligned}$$

It is easy to see that h is a bijection. □

Exercise Verify that h defined in the proof is indeed a bijection from A to B.

A function $f : \Sigma^* \to \Sigma^*$ is called *length increasing* if $|f(x)| > |x|$ for all $x \in \Sigma^*$. In order to adapt the Schröder-Bernstein proof strategy for showing polynomial-time isomorphisms between NP-complete sets, it turns out that length-increasing 1-invertible reductions is a suitable notion.

Theorem 3.3 [BH77] *Let $A, B \subseteq \Sigma^*$ be two languages such that there are length-increasing 1-invertible reductions from A to B and from B to A. Then A and B are polynomial-time isomorphic.*

The proof of this theorem is in the following exercise.

Exercise

1. Suppose f and g are length-increasing 1-invertible reductions from A to B and from B to A respectively. Show that the partition corresponding to infinite preimage sequences is empty for both f and g.
2. Verify that the bijection $h : \Sigma^* \to \Sigma^*$ defined in the proof of Theorem 3.2 is a polynomial-time isomorphism between A and B.

The question is how do we get hold of length increasing 1-invertible reductions? Berman and Hartmanis [BH77] discovered another natural property that several NP-complete languages are endowed with. A language $A \subseteq \Sigma^*$ is said to be *paddable* if there is a 1-invertible reduction *pad* from $A \times \Sigma^*$ to A. It turns out that many natural NP-complete problems are paddable.

Exercise Show that SAT, 3-SAT, CLIQUE, VC are all paddable languages.

Now, it turns out that paddable NP-complete languages are all isomorphic [BH77]. Since many NP-complete problems are paddable, it lead Berman and Hartmanis to make their conjecture.

Exercise

1. If $A \leq_m^p B$ and B is paddable then show that A is polynomial-time reducible to B via a 1-invertible length increasing function.
2. Conclude that if A and B are paddable NP-complete languages then they are polynomial-time isomorphic.

1.4 Are There Sparse NP-Complete Sets?

Let $A \subseteq \Sigma^*$ be any language. The *density* of A at length n is defined to be the number $|A^{=n}|$ of length n strings in A. The language A is exponentially dense if $|A^{=n}|$ is at least 2^{n^ϵ} for some constant $\epsilon > 0$. Natural NP-complete problems have exponential density.

Exercise Show that SAT, CLIQUE, VC have exponential density.

A consequence of the Berman-Hartmanis conjecture is that all NP-complete languages have exponential density. Can we show that languages with subexponential density cannot be NP-hard?

A language $A \subseteq \Sigma^*$ is said to be *sparse* if there is a polynomial $p(n)$ such that $|A^{=n}| \leq p(n)$ for all n.

Theorem 4.1 (Mahaney) [Mah82] *No sparse language is* NP*-hard unless* P = NP.

Proof We present a more recent proof [Agr11]. Suppose SAT $\leq_m^p A$ for some arbitrary sparse language A via a polynomial-time reduction f. For some polynomial $p(n)$ we have $|f(x)| \leq p(|x|)$.

We give a polynomial-time algorithm for SAT. Given a formula F of size s in boolean variables $x_1, x_2, \ldots, x_n$ as input, the algorithm proceeds in stages $0 \leq i \leq n$. At stage i, the algorithm maintains a list of formulas $\{F_a \mid a \in I\}$ such that each $F_a, a \in I$ is obtained from F by the truth assignment a to the first i variables $x_1, x_2, \ldots, x_i$ with the property that

$$F \in \text{SAT} \text{ if and only if } F_a \in \text{SAT} \text{ for some } a \in I. \tag{1.1}$$

If $|I| > q(s) + 1$, for a suitable polynomial $q(s)$ which will be defined in the course of the proof, then the algorithm applies a pruning operation to discard some a from the list I such that Property (1.1) holds for $I \setminus \{a\}$ as well. We now describe the pruning operation:

Consider all pairs of disjunctions $F_{ab} = F_a \vee F_b$ for $a, b \in I$. If s is the size of formula F then clearly $2s$ bounds the size of F_{ab} for all $a, b \in I$. Fix an $a \in I$ and consider $f(F_{ab})$ for all $b \in I \setminus \{a\}$. If $F_a \in \text{SAT}$ then clearly $F_{ab} \in \text{SAT}$ for all $b \in I \setminus \{a\}$. Hence, $\mathbb{F}_a \in \text{SAT}$ implies $f(F_{ab}) \in A^{\leq p(2s)}$. Since A is sparse we know that $|A^{\leq p(2s)}| \leq q(s)$ for some polynomial q. The algorithm computes $f(F_{ab})$ for all $b \in I \setminus \{a\}$ and does the following:

1. If $f(F_{ab})$ are all distinct for $b \in A \setminus \{a\}$ then F_a cannot be in SAT and a can be discarded from I.
2. Otherwise, for some $b \neq c \in I \setminus \{a\}$ we have $f(F_{ab}) = f(F_{ac})$. The algorithm can discard either b or c from I.

After pruning the list to get $\{F_a \mid a \in I\}$ such that $|I| \leq q(s) + 1$ the algorithm proceeds to the next stage where it first doubles the list of formulas by setting x_{i+1} to 0 and 1 to get $\{F_b \mid b \in J\}$ where J consists of all extensions of assignments in I obtained by setting x_{i+1} to 0 and 1.

Continuing thus, at the nth stage the algorithm accepts SAT if the set I at that stage includes a satisfying assignment. □

Let $A \in$ NP. By definition there are a polynomial-time computable language $B \subseteq \Sigma^* \times \Sigma^*$ and a polynomial bound $p(n)$ such that $x \in A$ if and only if

$$\exists y \in \Sigma^{p(|x|)} \langle x, y \rangle \in B.$$

Define the language $\text{pre}(A) = \{\langle x, w\rangle \mid \exists u \in \Sigma^{p(|x|)-|w|} : \langle x, wu\rangle \in B\}$.

Exercise Show that $\text{pre}(A)$ is in NP and $A \leq_m^p \text{pre}(A)$.

A *tally* language is a subset of 0^*. The next exercise is about tally languages and is analogous to Theorem 4.1.

Exercise For a language $A \in \text{NP}$ if $\text{pre}(A)$ is polynomial-time reducible to a tally language then show that $\text{pre}(A)$, and hence A, is in P.

1.4.1 Subexponentially Dense Languages

A subset $S \subseteq \Sigma^*$ is of *subexponential density* if for every $\epsilon > 0$ there is an $n_0 \in \mathbb{N}$ such that for all $n > n_0$ we have $|S^{=n}| \leq 2^{n^\epsilon}$. A natural question is whether Mahaney's theorem can be generalized to sets of subexponential density.

The class of languages $\text{SUBEXP} = \cap_{\epsilon>0}\text{DTIME}[2^{n^\epsilon}]$ consists of languages that have subexponential time decision procedures.

In the proof of Theorem 4.1 we gave a polynomial-time algorithm for SAT assuming that it is reducible to some sparse language. Modify the proof to show the following.

Exercise If SAT is polynomial-time many-one reducible to a language of subexponential density then show that $\text{NP} \subseteq \text{SUBEXP}$.

The inclusion $\text{NP} \subseteq \text{SUBEXP}$ is believed unlikely. For instance, it would imply a $2^{o(n)}$ time algorithm for the satisfiability of 3CNF formulas, where n is the number of boolean variables in the input formula. This would, in turn, imply similar subexponential algorithms for certain other NP-complete problems [IPZ01]. For a complexity theory tailored to the problem of designing faster exponential-time algorithms for NP-complete problems see [IPZ01] and related papers.

For the rest of this section we briefly discuss another unlikely complexity-theoretic consequence, assuming SAT is reducible to a set of subexponential density. In the process we will introduce some more basic complexity theory.

We start with the definition of the *polynomial-time hierarchy*. A language L is in the class Σ_k^p if there are a polynomial $p(n)$ and language $A \in \text{P}$ such that

$$L = \{x \mid \exists y_1 \forall y_2 \cdots \text{Q} y_k : |y_i| \leq p(|x|) \text{ for all } i \text{ and } \langle x, y_1, y_2, \ldots, y_k\rangle \in A\},$$

where the quantifier "Q" is "$\exists$" if k is odd and "$\forall$" if k is even. The class Π_k^p consists of all languages L such that $\Sigma^* \setminus L$ is in Σ_k^p. The following observations are immediate from the definition: