# Complexity Theory

## Introduction and Motivation

Daniel Borchmann, Markus Krötzsch

Computational Logic

2015-10-14

# Introduction and Organisation

# Course Tutors

Markus Krötzsch
Lectures

Daniel Borchmann
Lectures and Exercises

# Organization

## Lectures

Tuesday, DS 2 (9:20–10:50), APB E005
Wednesday, DS 4 (13:00–14:30), APB E005

## Exercise Sessions (starting 23 October)

Friday, DS 4 (13:00–14:30), APB E005

## Web Page

`https://ddll.inf.tu-dresden.de/web/Complexity_Theory_(WS2015)`

## Lecture Notes

Slides of current and past lectures will be online. The are **no** notes for blackboard lectures.

# Goals and Prerequisites

## Goals

- Introduce basic notions of *computational complexity theory*
- Introduce commonly known complexity classes (P, NP, PSpace, . . . ) and discuss relationships between them
- Develop tools to classify problems into their corresponding complexity classes
- Introduce (some) advanced topics of complexity theory

## (Non-)Prerequisites

- No particular prior courses needed
- General mathematical and theoretical computer science skills necessary

# Reading List

- **Michael Sipser: *Introduction to the Theory of Computation, International Edition*; 3rd Edition; Cengage Learning 2013**
- Sanjeev Arora and Boaz Barak: *Computational Complexity: A Modern Approach*; Cambridge University Press 2009
- Michael R. Garey and David S. Johnson: *Computers and Intractability*; Bell Telephone Laboratories, Inc. 1979
- Erich Grädel: *Complexity Theory*; Lecture Notes, Winter Term 2009/10
- John E. Hopcroft and Jeffrey D. Ullman: *Introduction to Automata Theory, Languages, and Computation*; Addison Wesley Publishing Company 1979
- Neil Immerman: *Descriptive Complexity*; Springer Verlag 1999
- Christos H. Papadimitriou: *Computational Complexity*; 1995 Addison-Wesley Publishing Company, Inc

# Motivation

# Computational Problems are Everywhere

## Example 1.1

- What are the factors of 54,623?
- What is the shortest route by car from Berlin to Hamburg?
- My program now runs for two weeks. Will it ever stop?
- Is this C++ program syntactically correct?

## Clear

Computational Problems are ubiquitous in our everyday life!
And, depending on what we want to do, those problems either need to be *easily solvable* or *hardly solvable*.

Approach to problems:

> *[T]he way is to avoid what is strong, and strike at what is weak.*

(Sun Zi: The Art of War, Chapter 6: Weak Points and Strong)

# Examples

## Example 1.2 (Shortest Path Problem)

Given a weighted graph and two vertices $s$, $t$, find the shortest path between $s$ and $t$.

Easily solvable using, e.g., Dijkstra's Algorithm.

## Example 1.3 (Longest Path Problem)

Given a weighted graph and two vertices $s$, $t$, find the *longest* path between $s$ and $t$.

No efficient algorithm known, and believed to not exist.
(i.e., this problem is *NP-hard*)

## Observation

Difficulty of a problem is hard to assess

# Measuring the Difficulty of Problems

## Question

How can we measure the complexity of a problem?

## Approach

Estimate the resource requirements of the "best" algorithm that solves this problem.

Typical Resources:

- Running Time
- Memory Used

## Note

To assess the complexity of a problem, we need to consider *all possible algorithms* that solve this problem.

# Problems

## What actually is . . . a Problem?

(Decision) Problems are *word problems* of particular languages.

## Example 1.4

"Problem: Is a given graph connected?" will be modeled as the word problem of the language

$$\text{GCONN} := \{ \langle G \rangle \mid G \text{ is a connected graph} \}.$$

Then for a graph $G$ we have

$$G \text{ is connected} \iff \langle G \rangle \in \text{GCONN}.$$

## Note

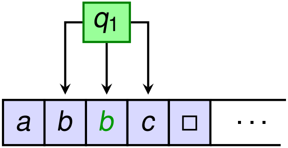
The notation $\langle G \rangle$ denotes a suitable "encoding" of the graph $G$ over some fixed alphabet (e.g., $\{ 0, 1 \}$).

# Algorithms

## What actually is ... an Algorithm?

Different approaches to formalize the notion of an “algorithm”

- Turing Machines
- Lambda Calculus
- $\mu$-Recursion
- ...



# Avoid What is Strong

Suppose we are give a language $\mathcal{L}$ and a word *w*.

### Question

Does there need to exist *any* algorithm that decides whether $w \in \mathcal{L}$?

### Answer

No. Some problems are *undecidable*.

### Example 1.5

- The Halting Problem of Turing machines
- The Entscheidungsproblem (Is a mathematical statement true?)
- Finding the lowest air fare between two cities (→ Reference)
- Deciding syntactic validity of C++ programs (→ Reference)

Avoid: Suppose from now on all problems we consider to be decidable.

# Time and Space

## Drawback

Measuring running time and memory requirements depends highly on the *machine*, and not so much on the *problem*.

## Resort

Measure time and space only *asymptotically* using *Big-$\mathcal{O}$*-Notation:

$$f(n) = \mathcal{O}(g(n)) \iff f(n) \text{ “asymptotically bounded by” } g(n)$$

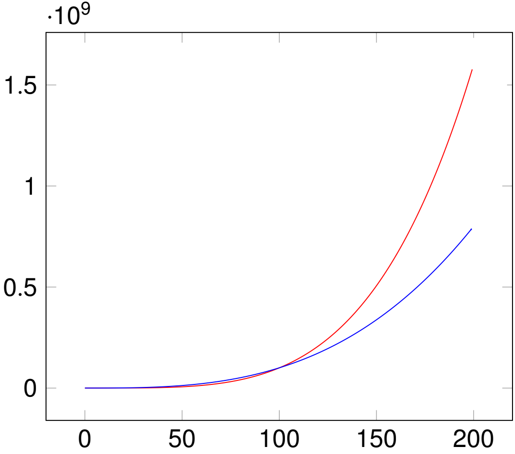
More formally:

$$f(n) = \mathcal{O}(g(n)) \iff \exists c > 0\, \exists n_0 \in \mathbb{N}\, \forall n > n_0 : f(n) \leq c \cdot g(n).$$

# Big-$\mathcal{O}$-Notation

Example 1.6

$100n^3 + 1729n = \mathcal{O}(n^4)$:



# Complexity of Problems

## Approach

The *time (space) complexity* of a problem is the asymptotic running time of a fastest (least memory consumptive) algorithm that solves the problem.

## Problem

Still too difficult . . .

## Example 1.7 (Traveling Salesman Problem)

Given a weighted graph, find the shortest simple path visiting every node.

- Best known algorithm runs in time $\mathcal{O}(n^2 2^n)$ (Bellman-Held-Karp algorithm)
- Best known lower bound is $\mathcal{O}(n \log n)$

Exact complexity of TSP unknown.

# Even more abstraction

## Approach

Divide decision problems into the “quality” of their fastest algorithms:

- P is the class of problems *solvable in polynomial time*
- PSpace is the class of problems *solvable in polynomial space*
- ExpTime is the class of problems *solvable in exponential time*
- L is the class of problems *solvable in logarithmic space* (apart from the input)
- NP is the class of problems *verifiable in polynomial time*
- NL is the class of problems *verifiable in logarithmic space*

And many more!

$\oplus$P, #P, AC, $\mathrm{AC}^0$, ACC0, AM, AP, APSpace, BPL, BPP, BQP, coNP, E, Exp, FP, IP, MA, MIP, NC, NExpTime, P/poly, PH, PP, PSpace, RL, RP, $\Sigma_i^\mathrm{p}$, TISP$(T(n), S(n))$, ZPP, ...

# Strike at What is Weak

## Approach (cf. Cobham-Edmonds-Thesis)

The problems in P are “tractable” or “efficiently solvable” (and those outside not)

## Example 1.8

The following problems are in P:

- Shorts Path Problem
- Satisfiability of Horn-Formulas
- Linear Programming
- Primality

## Note

The Cobham-Edmonds-Thesis is only a *rule of thumb*: there are (practically) tractable problems outside of P, and (practically) intractable problems in P.

# Friend or Foe?

### Caveat

It is not known how big P is.
In particular, it is unknown whether P $\neq$ NP or not.

### Approach

Try to find out which problems in a class are at least as hard as others.
*Complete* problems are then the hardest problems of a class.

### Example 1.9

Satisfiability of propositional formulas is *NP-complete*: if we can efficiently decide whether a propositional formula is satisfiable, we can solve *any* problem in NP efficiently.

But: we still do not know whether we can or cannot solve satisfiability efficiently. We only know it will be difficult to find out . . .

# Learning Goals

- Get an overview over the foundations of Complexity Theory
- Gain insights into advanced techniques and results in Complexity Theory
- Understand what it means to "compute" something, and what the strengths and limits of different computing approaches are
- Get a feeling of how hard certain problems are, and where this hardness comes from
- Appreciate how very little we actually know about the computational complexity of many problems

# Lecture Outline I

- **Turing Machines** (Revision)
  Definition of Turing Machines; Variants; Computational Equivalence; Decidability and Recognizability; Enumeration
- **Undecidability** (Daniel)
  Examples of Undecidable Problems; Mapping Reductions; Rice's Theorem (both for characterizing Decidability and Recognizability); Recursion Theorem; Outlook into Decidability in Logic
- **Time Complexity** (Markus)
  Measuring Time Complexity; Many-One Reductions; Cook-Levin Theorem; Time Complexity Classes (P, NP, ExpTime); NP-completeness; pseudo-NP-complete problems
- **Space Complexity** (Markus)
  Space Complexity Classes (PSpace, L, NL); Savitch's Theorem; PSpace-completeness; NL-completeness; NL $=$ coNL

# Lecture Outline II

- **Diagonalization** (Daniel)
  Hierarchy Theorems (det. Time, non-det. Time, Space); Gap Theorem; Ladner's Theorem; Relativization; Baker-Gill-Solovay Theorem
- **Alternation** (Markus)
  Alternating Turing Machines; APTime = PSpace; APSpace = ExpTime; Polynomial Hierarchy; $\text{NTIME}(n) \nsubseteq \text{TISP}(n^{1.2}, n^{0.2})$
- **Circuit Complexity** (Markus)
  Boolean Circuits; Alternative Proof of Cook-Levin Theorem; Parallel Computation (NC); P-completeness; P/poly; (Karp-Lipton Theorem, Meyer's Theorem)
- **Probabilistic Computation** (Daniel)
  Randomized Complexity Classes (RP, PP, BPP, ZPP); Sipser-Gács-Lautemann Theorem

# Lecture Outline III

- **Interactive Proofs** (Daniel)
  Prover and Verifier; Deterministic Proof Systems; Probabilistic Verifiers; The Class IP; IP $=$ PSpace; (Arthur-Merlin Proofs); (NP-completeness of Graph Isomorphism implies the Collapse of PH)
- **Descriptive Complexity** (Markus)
  Logical Descriptions of Complexity Classes; Fagin's Theorem; . . .
- **Approximation Complexity** (Daniel)
  Optimization Problems; Polynomial-Time Approximation Schemes; Reductions; Hardness of Approximations; (PCP-Theorem)
- **Cryptography** (Overview Only; Daniel)
  One-Way Functions; Pseudorandom Generators; Zero Knowledge

# Avoid what is Strong, and Strike at what is Weak

Sometimes the best way to solve a problem is to avoid it . . .