Algorithm Graph Theory:
How hard is your
combinatorial optimization problem?

Short Course – Lecture 1

June 7, 2017

Contents

1. Basic Complexity Theory and polynomial solvable graph classes:
   - P, NP, weak and strong NP-hardness
   - polynomial algorithms for combinatorial problems on interval, chordal, perfect, and perfect graphs
2. Graphs of bounded Treewidth:
   - path and tree decompositions of graphs
   - path- and treewidth
   - computing treewidth
   - dynamic programming algorithms for combinatorial problems on graphs of bounded treewidth

Contents

3. Fixed Parameter and Exact Algorithms:
   - fixed parameter tractability
   - kernelization
   - W-hierarchy
   - exponential time algorithms
   - branching algorithms, dynamic programming algorithms

Schedule

	9.00-10.30	10.30-11.00	11:00-12:30
Wed 06/07	**Basics:** Complexity	Break	**Basics:** First Examples
Thu 06/08	**Basics:** Interval graphs	Break	**Basics:** chordal and perfect graphs
Fri 06/09	**Treewidth:** introduction	Break	**Treewidth:** Graph classes of bounded treewidth
Mon 06/12	**Treewidth:** Lower and Upper Bounds	Break	**Treewidth:** Dynamic Programming
Tue 06/13	**FPT:** Parameterized Complexity	Break	**FPT:** Kernelization
Wed 06/14	**Exact:** Branching Algorithms	Break	**Exact:** Dynamic Programming

Appetizer

- Knapsack Problem
- Travelling Salesman Problem

A *very informal* introduction to computational complexity

Problems, Instances & Solutions

- A **problem** is a general question, where several parameters are left open
- A **solution** consist of answers for these parameters
- A problem is defined by a description of all its parameters and which properties require an answer
  - *Given a digraph D=(V,A), distances d(a), a source s and a target t, what is the length of a shortest path from s to t?*
- A **problem instance** is a specific input where all parameters are given explicitedly
  - *Let D be the road network of Clemson, distances given by travel times, s=211 Fernow St and t=581 Berkeley Dr.*
  - *How long does it take to go from s to t?*

Example

- The task „*Find a shortest traveling salesman tour in a graph!*“ is a problem with parameters a number of cities and a distance matrix
- The file „bier127.tsp“ is a problem instance

Algorithms & Efficiency

- We denote a problem by Π, whereas an instance of Problem Π is denoted by $I \in \Pi$
- An **algorithm** solves problem Π if for every problem instance $I \in \Pi$, the algorithm finds a solution
  - Dijkstra's algorithm finds a shortest path in any digraph with nonnegative distances, arbitrary source s and arbitrary target t
- The **aim** of designing algorithms is to develop **efficient** procedures to find a solution, where efficient refers to **time** and **memory storage**

Two Problem Types

- **Decision problems:** Problems that can be answered by **„yes“ or „no“**
  - „Does there exist a solution to the TSP with value at most K?“ can be answered by „yes“ or „no“
- **Optimization problems:** Problems that ask to find an object with certain prescribed properties
  - „What ist the shortest traveling salesman tour in this graph?“ requires to provide a tour
- **Of course**, the answer „yes“ should be verifyable with a tour of length at most K, and an answer „no“ should be guaranteed as well
- For yes/no decision problems, we do not have to distinguish between **solution** and **optimal solution**; for optimization problems we do.

Problem Encoding

- The **time complexity** (resp. **memory complexity**) of an algorithm depends in general on the „size“ of the problem instance, i.e., the amount of input data.
- The **encoding** of a problem instance is of critical importance
- Integers are *binary encoded*:
  - Nonnegative integer n requires $\lceil \log_2(n+1) \rceil$ bits
  - One more bit is required for the sign of an integer
- The **coding length** $\langle I \rangle$ of an instance $I \in \Pi$ is the number of bits required to encode I completely

Problem Encoding

- The coding length of
  - an integer n is $\langle n\rangle := \lceil \log_2(n+1) \rceil + 1$
  - a rational r=p/q is $\langle r\rangle := \langle p\rangle + \langle q\rangle$
  - a vector $x=(x_1,\ldots,x_n)^T \in Q^n$ is $\langle x\rangle := \sum \langle x_i\rangle$
  - a matrix $A \in Q^{m \times n}$ is $\langle A\rangle := \sum \sum \langle a_{ij}\rangle$
- A (simple) graph with n vertices and m edges can be encoded in different ways:
  - vertex-edge incident matrix
  - adjacency list for every vertex
  - vertex-vertex incident matrix

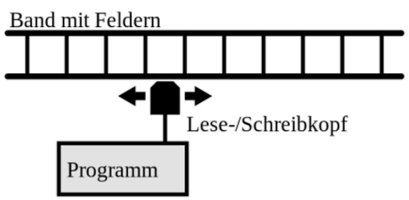
Example: Knapsack

- Input consists of vectors $c \in Q^k$ and $a \in Q^k$, scalar $b \in Q$
- Knapsack with k items has input length
  $\langle I \rangle = \langle c \rangle + \langle a \rangle + \langle b \rangle \leq 2k * \langle \max \{c_i, a_i\} \rangle + \langle b \rangle$
  $\leq (2k+1) * \langle b \rangle$

CLEMSON UNIVERSITY RWTHAACHEN UNIVERSITY

Computing Model

- To execute an algorithm and to compute its running time and memory requirement depending on the input length of a problem instance, we need a **computing model**
- Examples:
  - Turing machine
  - RAM machine



- An algorithm first reads the data of a problem instance and uses for this <I> bits of memory
- Further bits are required to compute the solution

Memory requirement

- The number of bits that are used *at least once* during the execution of Algorithm A is called the **memory requirement** of **A to solve I**

- Example:
  - Dynamic Programming for Knapsack with k items requires at most k*b*<C> bits of memory where C is Σc_i
  - Or b*<C> bits of memory if memory is reused

- The memory requirement is estimated from above

Running time

- The **running time of A to solve I** is the number of elementary operations which A requires until the end of the procedure.
- Elementary operations are
  - Reading, writing, and deleting,
  - Addition, subtraction, multiplication, division and comparison
- of rational (or integer) numbers.
- Here, we estimate each operation w.r.t. the maximum numbers involved

A bit more formally

The function $f_A : \mathbb{N} \to \mathbb{N}$ **defined by**

$$f_A(n) := \max_{I \in \Pi \textbf{ with } <I> \leq n} \{ \textbf{ running time of } A \textbf{ to solve } I \}$$

is called the running time function of A.

The function $s_A : \mathbb{N} \to \mathbb{N}$ **defined by**

$$s_A(n) := \max_{I \in \Pi \textbf{ with } <I> \leq n} \{ \textbf{ memory requirement of } A \textbf{ to solve } I \}$$

is called the memory function of A.

The algorithm A **has polynomial running time (short:** A **is a polynomial algorithm) if there exists a polynomial** $p : \mathbb{N} \to \mathbb{N}$ **with** $f_A(n) \leq p(n)$ **for all** $n \in N$**.**
If p **is a polynomial of degree** k**, we call** f_A **of order at most** n^k **and write** $f_A = \mathcal{O}(n^k)$

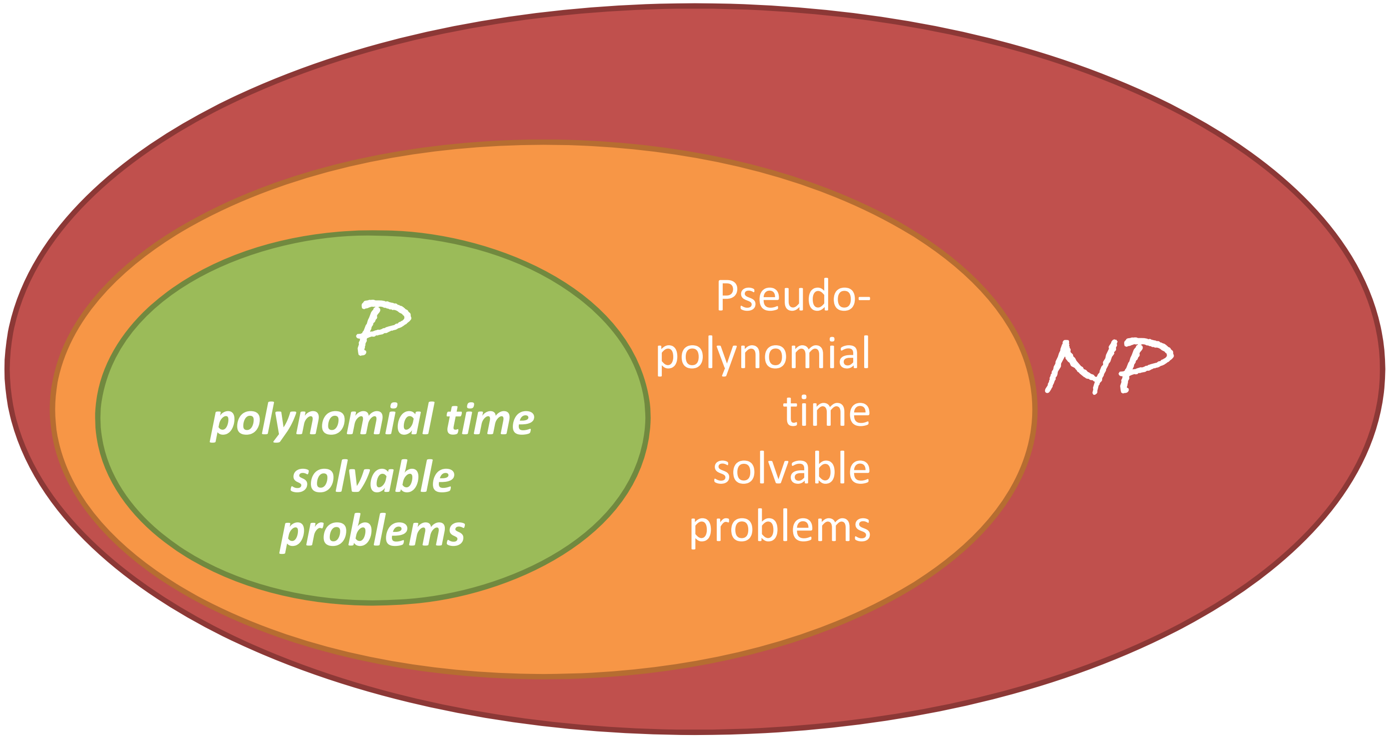
Algorithm A **has polynomial memory requirements if there exists a polynomial** $q : \mathbb{N} \to \mathbb{N}$ **with** $s_A(n) \leq q(n)$ **for all** $n \in N$**.**

Pseudopolynomiality

Algorithm A has pseudopolynomial running time if the running time is bounded by a polynomial p in both $< I >$ and the values of the input data.

Algorithm A has pseudopolynomial memory requirements if the memory consumption is bounded by a polynomial q in both $< I >$ and the values of the input data.

Classes of Problems



The class of all decision problems for which there exists a polynomial time algorithm is denoted by $\mathcal{P}$

A decision problem Π belongs to the class $\mathcal{NP}$ (nondeterministic polynomial) if

a) For every problem instance $I \in \Pi$ with positive answer an object Q exists which allows ist verification

b) There exists an algorithm taking problem instance I and Q as input to verify on the basis of Q the positive answer, which runs polynomial in <I>

Example

Are the following problems part of $\mathcal{NP}$?

- Does graph G have a cycle?
  - Q =
- Does graph G have a Hamilton cycle?
  - Q =
- Does G not have a Hamilton cycle?

- $co\text{-}\mathcal{NP}$ is the class of problems, where a negative answer can be verified in polynomial time (with an object Q).

CLEMSON UNIVERSITY RWTH AACHEN UNIVERSITY

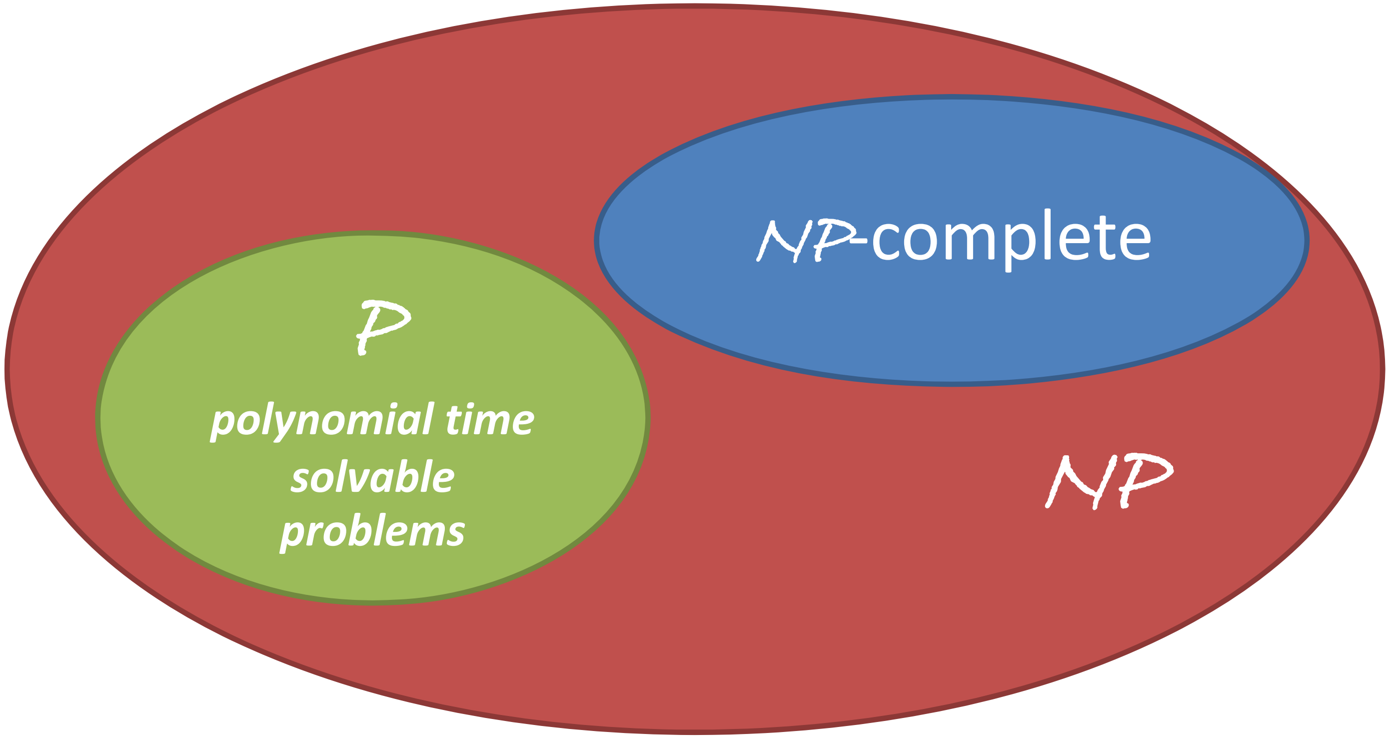
Results & Questions

- $P \subseteq NP$
- $P \subseteq \text{co-}NP$
- $P \subseteq (NP \cap \text{co-}NP)$
- Question: P=NP ?

Polynomial transformation

- Let Π_1 and Π_2 be two decision problems. A polynomial transformation of Π_1 to Π_2 is a polynomial time algorithm which constructs from a problem instance $I_1 \in \Pi_1$ a problem instance $I_2 \in \Pi_2$ such that the answer of I_1 is positive if and only if the answer of I_2 is positive.

- Remark: if Π_2 is solvable in polynomial time, then also Π_1

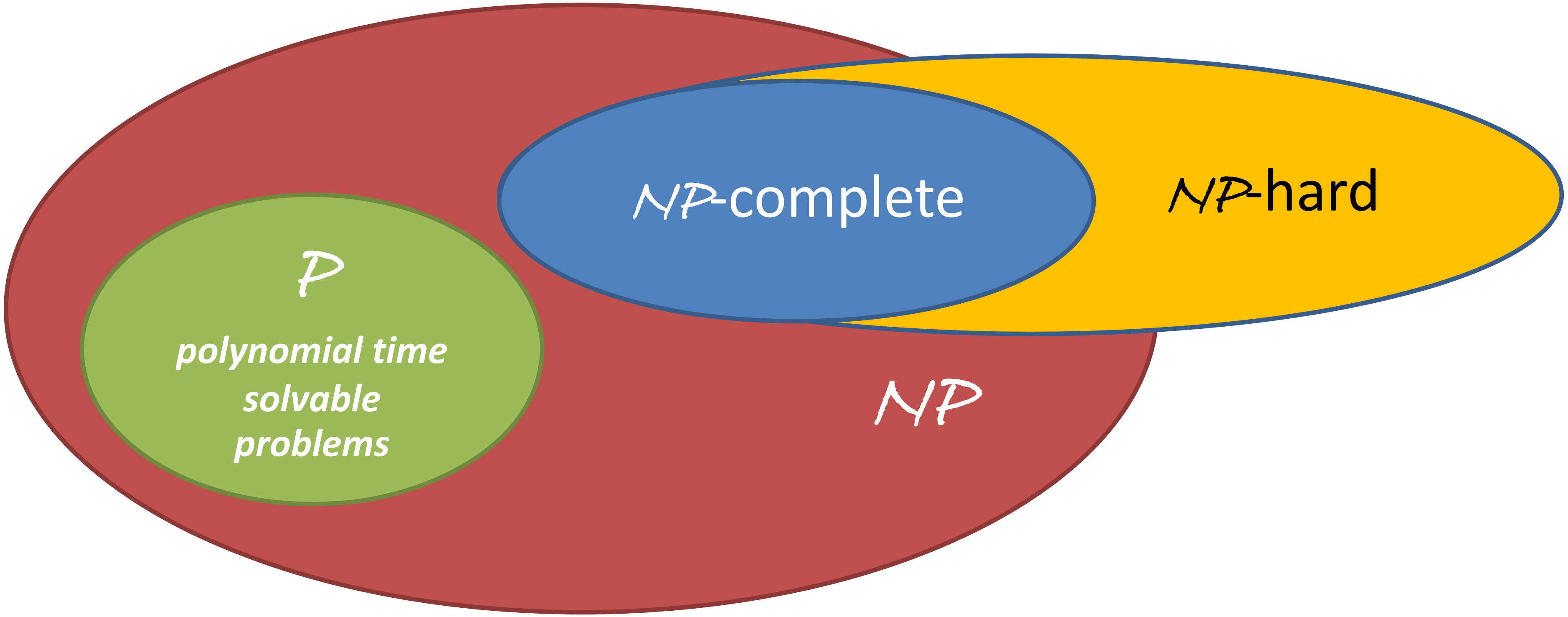


- A decision problem Π is called NP-complete if $\Pi \in$NP and every other problem in NP can be polynomial transformed to Π.
- Remark: if any NP-complete problem can be solved in polynomial time, all can, i.e., P=NP

Examples

- SAT is NP-complete
- K-SAT is NP-complete
- EXACT COVER is NP-complete
- DIRECTED HAMILTON CYCLE is NP-complete
- UNDIRECTED HAMILTON CYCLE is NP-complete

- TSP is NP-complete



- A problem is NP-hard if all problems in NP can be polynomially transformed to it (but it is not necessarily known whether it is in NP)
- Remark: Optimization versions of NP-complete problems are NP-hard: