• mailing list!

DAA2

What will we do?
- What does it mean to prove something hard/intractable?
- How to — // — // — → „master techniques" – key problems – proof styles – gadgets
- NOT a pure complexity course
- Approximation algorithms

- Online algorithms

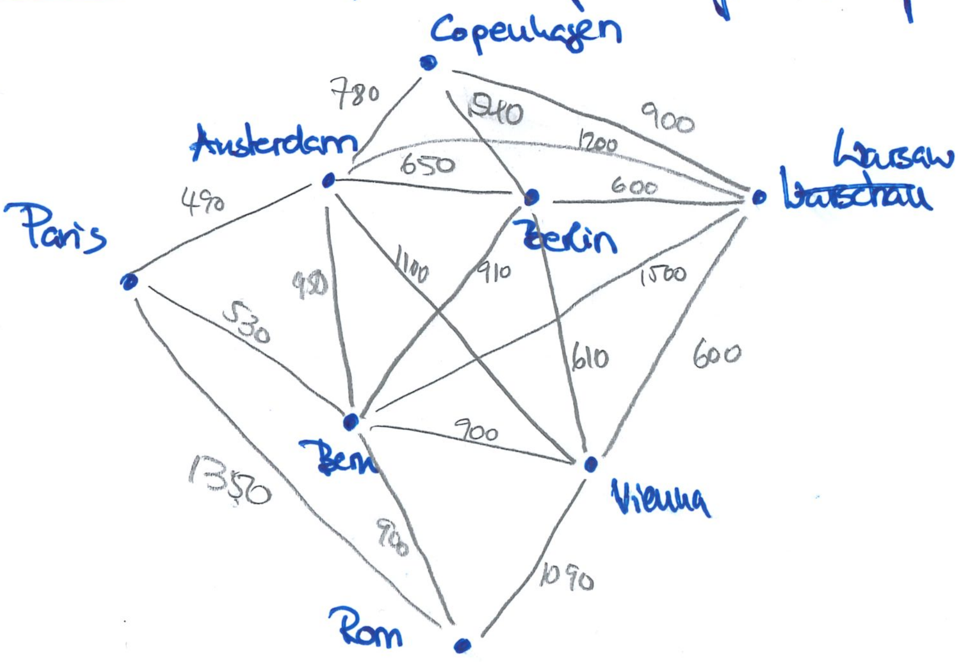
Literature:
- Garey, Johnson: Computers and Intractability; A Guide to the Theory of NP-completeness
- V. Vazirani: Approximation Algorithms
- Borodin / El-Yaniv: Online Computation and Competitive Analysis
- Course by Erik Demaine

1. Problems, Instances ~~etc~~, P, NP, EXP, hardness – Notation

* A problem Π gives a general description of parameters and statements of properties of a solution.

  ↳ example: TSP
  - cities ⟷ vertices $(v_1, \dots, v_n)$
  - roads ⟷ edges $(e_1, \dots, e_m)$
  - distances ⟷ weights $(c_{e_1}, \dots, c_{e_m})$
  - We ask for a shortest tour that is a round trip visiting each vertex exactly once

* An instance $I \in \Pi$ define values for all problem parameters



* The <u>input size</u> of instance $I \in \Pi$: # of symbols of the underlying alphabet necessary to encode I

What are we looking for?

↳ Algorithm to solve our problem (instances)!

↳ any algorithm?

* An algorithm solves a problem Π in <u>polynomial time</u> (n^c) if we can bound its running time for any $I \in \Pi$ by a polynomial of I's input size.

Can we always find this? Probably not! (unless P≠NP... more later)

Some more notation:

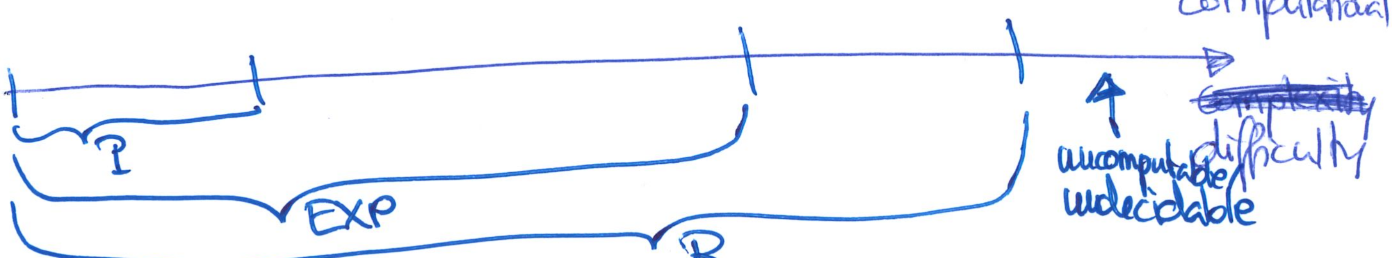
* A <u>decision problem</u> that allows for a polynomial time alg. belongs to <u>P</u>.

↳ answer is YES or NO
but: the TSP is an optimization problem?
We still use the same notation: give a numerical parameter B as a parameter
↳ Is there a tour of length less or equal to B?

P = { problems solvable in polynomial time }

EXP = { problems solvable in <u>exponential</u> time } ↳ 2^{n^c}

R = { problems solvable in finite time } [Turing 1936; Church 1941]
↳ „recursive"



$\boxed{P \subsetneq EXP \subsetneq R}$

Examples:

- negative-weight cycle detection $\in P$
- $n \times n$ Chess $\in EXP$ but $\notin P$
  - ∠ = who wins from given board configuration
- Tetris $\in EXP$, but not known whether $\in P$
  - ∠ = survive given pieces from given board
- halting problem $\notin R$ [Turing, 1936] → Turing machine
  - ∠ decide: given: computer program + input
    - ↔ will it halt or run forever?
- „Most" decision problems $\notin R$
  (# algorithms $\approx \mathbb{N}$; # decision problems $\approx 2^{\mathbb{N}} = \mathbb{R}$)

* A problem belongs to NP if for every „YES"-instance there is a certificate in polynomial time.

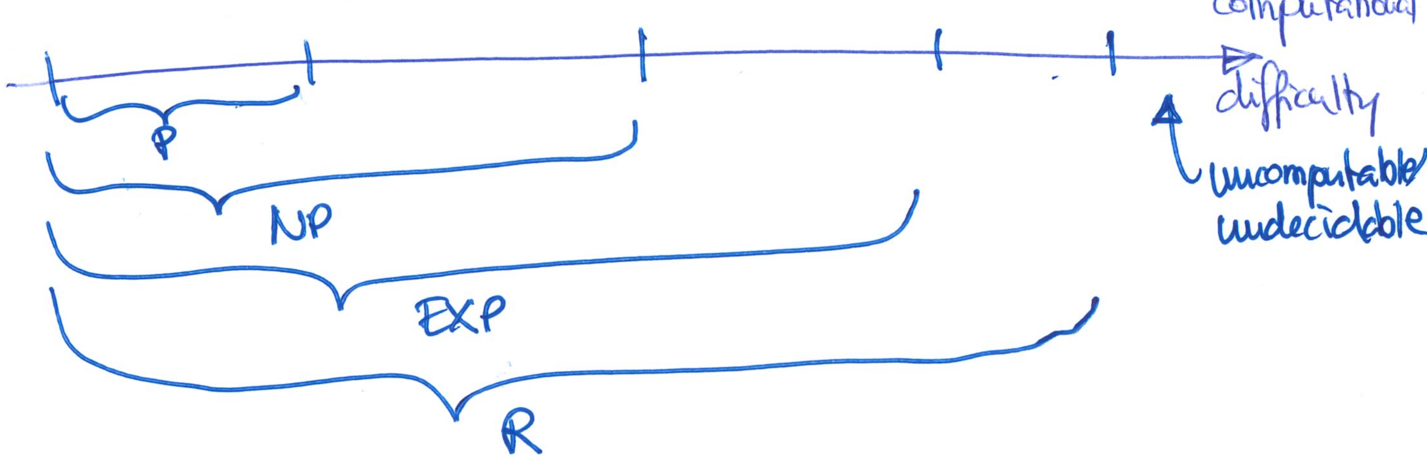
NP = {decision problems solvable in polytime with a „lucky" algorithm}

∠ can make lucky guesses, always „right" w/o trying all options

- Nondeterministic model: algorithm makes guesses and then says YES or NO
- guesses guaranteed to lead to YES outcome if possible (NO o.w.)

= {decision problems with solutions that can be „checked" in polynomial time}

when answer = YES can „prove" it + polytime alg. can check proof



Example: Tetris $\in NP$
- nondeterministic alg: - guess each move - did I survive?
- proof of YES: list what moves to make (rules of Tetris easy)

$\boxed{P \neq NP}$: big conjecture (worth \$ 1.000.000)

≈ can't engineer luck

≈ generating (proofs of) solutions can be harder than checking them

* A problem Π is <u>NP-hard</u> if $\Pi \in P$ implies $P = NP$.
* In general: <u>X-hard</u> = "as hard as" every problem $\in X$

  ↳ NP, EXP, etc

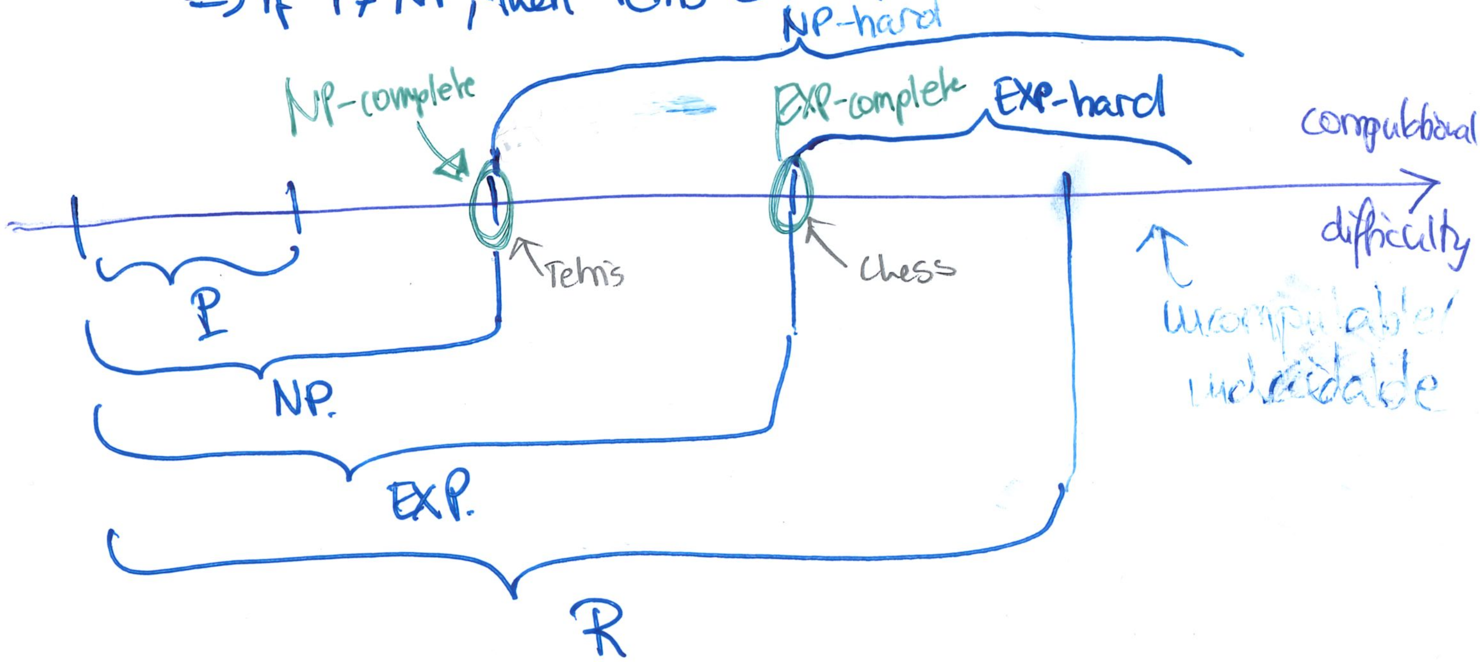
* A problem Π is <u>NP-complete</u> if it is NP-hard and in NP.
* <u>X-complete</u> = X-hard $\cap$ X

what does that mean

↳ a bit later

<u>Example</u>: Tetris is NP-complete [Breukelaar et al., 2004]

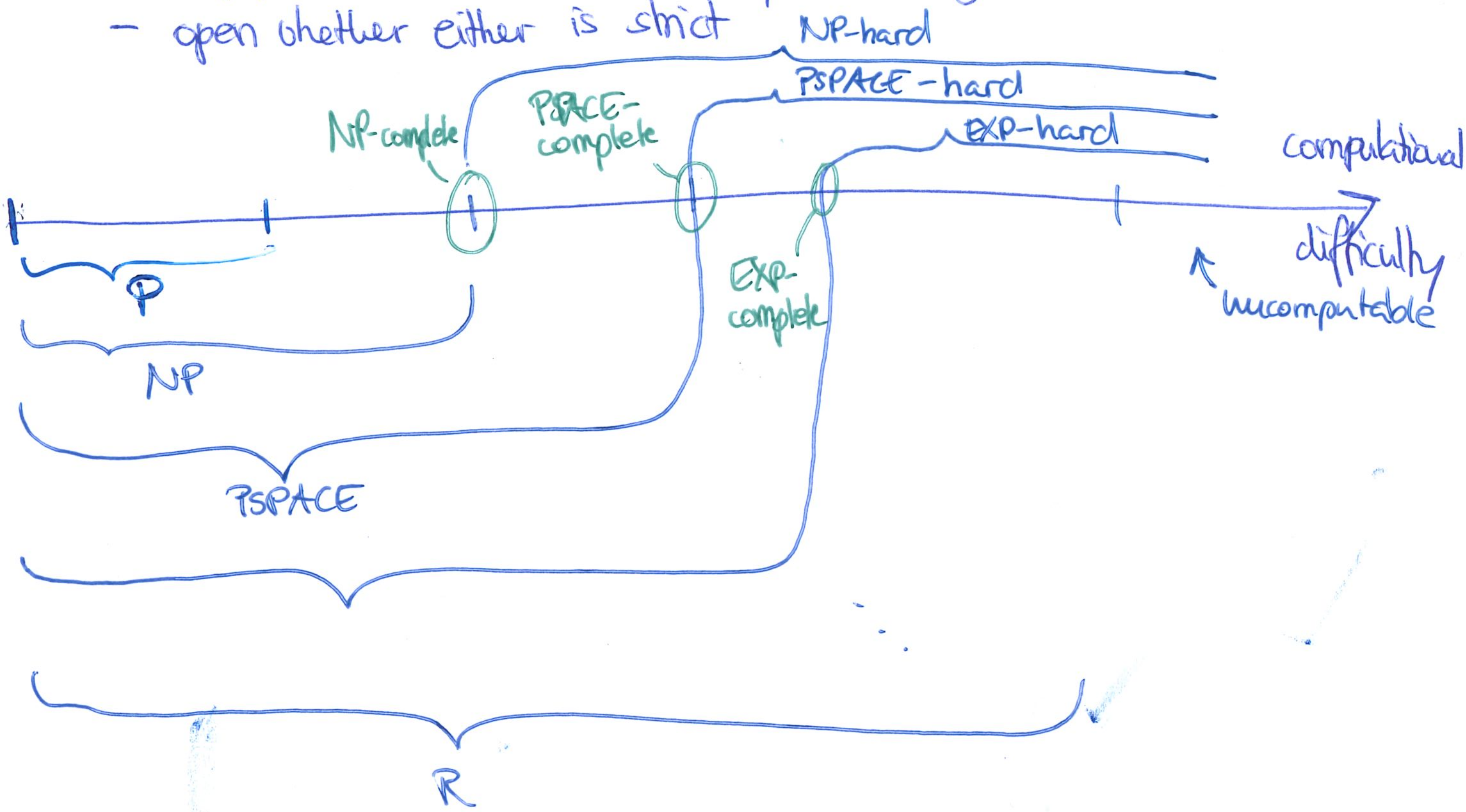
$\Rightarrow$ if $P \neq NP$, then Tetris $\in NP \setminus P$



Chess is EXP-complete $\Rightarrow \notin P$

$\Rightarrow$ Chess $\in EXP \setminus NP$ if $NP \neq EXP$ (also open

* PSPACE = {problems soluable in polynomial space}
  - ⊆ EXP: only exp. many states
  - ⊇ NP: simulate all executions, take running OR
  - open whether either is strict



Example: RushHour is PSPACE-complete [Flake & Baum, 2002]

* There exists stuff beyond exponential – not so important, and not for this course

2. How to prove a problem to be NP-hard? What does "as hard as" mean?

What does it mean:

~~* No known NP-har~~

Reduction from problem A to problem B = poly-time algorithm to convert: instance of A → instance of B

such that solution to A = solution to B

⇒ if can solve B then can solve A

⇒ B is at least as hard as A

(A is a special case of B)

* E.g.: no NP-complete problem can be solved by any known polytime alg.
* If there is a polytime alg. for any NP-complete problem, then there are polytime alg.s for all NP-complete problems

→ Interest in approximation algorithms!
„If I cannot give an optimal solution in polytime, I like to give guarantees on how „bad" the solutions an algorithm gives can be — compared to the (unknown!) optimum" → interest in bounds

Proof of NP-hardness

Our problem: $\boxed{B}$

Some known NP-complete problem: A

Reduction? Which direction?

Idea: „dress up" every instance of A as an instance of B

instance of B: $\tilde{I}$

If we had a polytime algorithm for B, we could solve $\tilde{I}$

→ This would give us the correct answer for I

Ensure with reduction!

→ We would have a polytime algorithm for A

reduction: every instance of $A \in NP$: input x

↓ instance of B

↓ input T(x) such that answer (YES/NO) for T(x) as an answer for B is the correct answer for x as an input for A

- This is a „one-call" reduction [Karp]
- „Multi-call" reduction [Turing] also possible: solve A using an oracle that solves B (doesn't help much for problems we consider)

Almost all hardness proofs are by reduction from known hard problem to your problem.

3. ~~Problems useful for reductions~~

Summary of recipe to show NP-completeness

(1) Show B is in NP
(2) Select a known NP-complete problem ~~B~~ A
(3) Construct a reduction from A to B
(4) Prove that reduction is polynomial

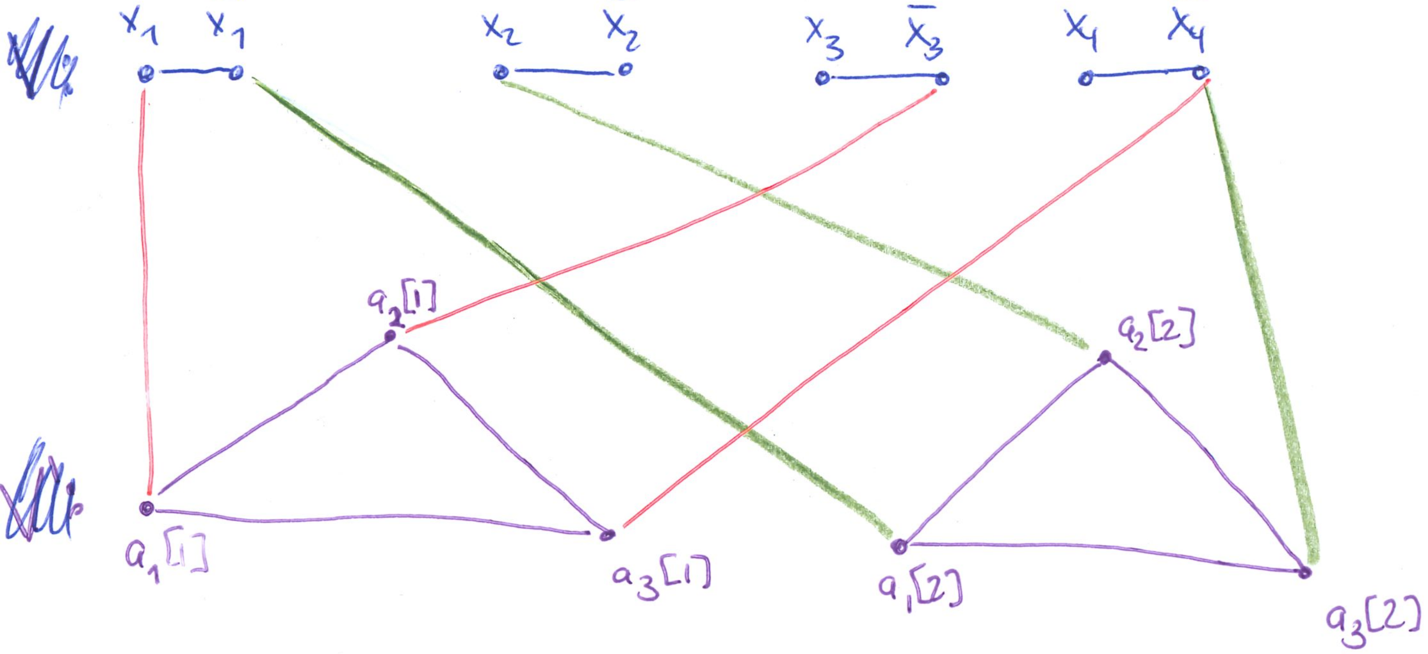
ad proof of Th. 3.1

$X = \{x_1, x_2, x_3, x_4\}$

also used $\bar{x}_1$

~~$C = \{\{x_1, x_3$~~

$$C = (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$$
$$= (x_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$$
$$= \{\{x_1, \bar{x}_3, \bar{x}_4\}, \{\bar{x}_1, x_2, \bar{x}_4\}\}$$



ad number problems:

Assuming $P \neq NP$: