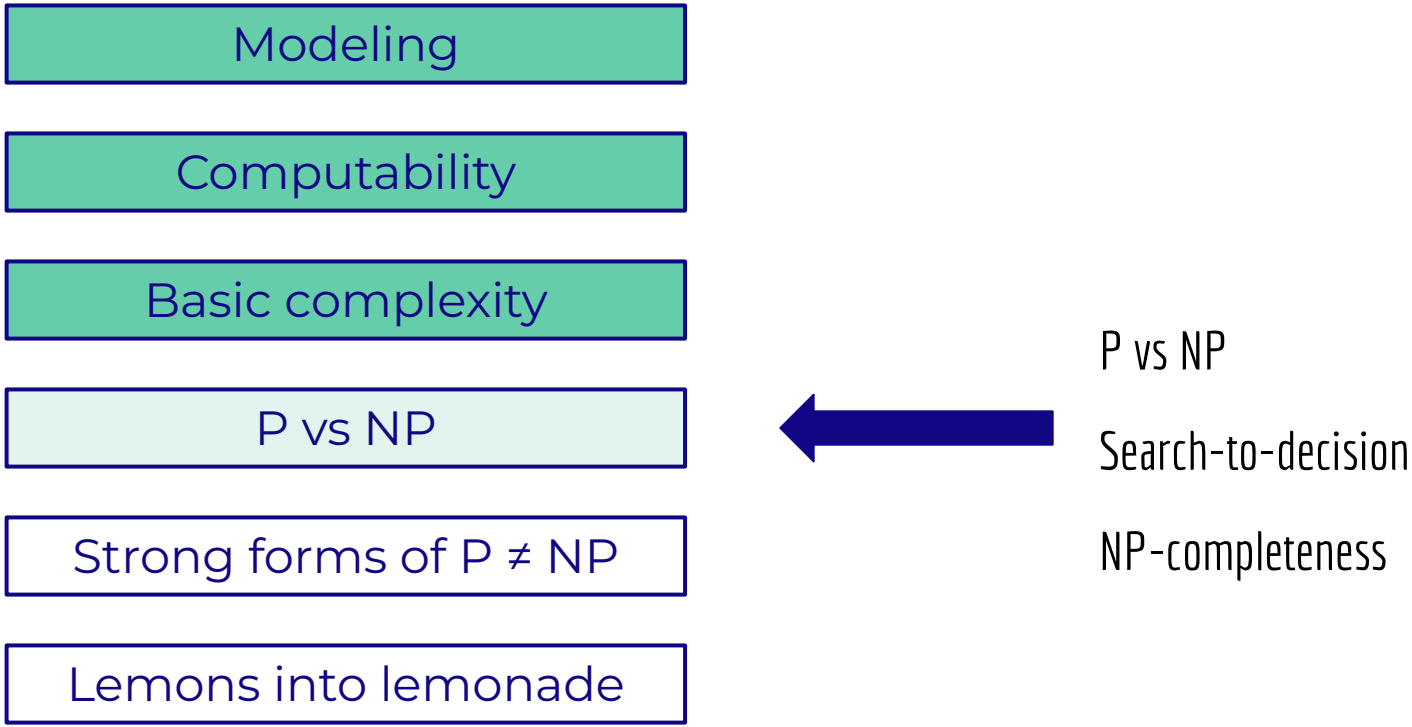
Complexity Theory

Instructor: Roei Tell

P vs NP

Course outline



Efficiently computable problems

- Definition:

  $\mathbf{P} = U_c \, TIME[n^c]$

- All languages decidable in polynomial time.

Efficiently computable problems

› Lower bounds: What lies outside $\mathbf{P}$?

Efficiently computable problems

- Lower bounds: What lies outside $\mathbf{P}$?
  - Uncomputable problems
    - DIAG, HALT, U, K, …
  - Hierarchies (more resources ⇒ more power)
    - time-bounded DIAG, U_T, …
  - How about even more natural, everyday problems?

Efficiently verifiable problems

- Intuition:

  Many problems seem hard to solve, but we can verify a solution if somebody gives it to us.

Efficiently verifiable problems

- Examples:
  - math: proving theorems, satisfying first-order logic ("∃")
  - games: sudoku, Rubik's cube, Tetris
  - graphs: Hamiltonian paths, k-size clique
  - arithmetic: factoring, solving a system of equations
  - …

Efficiently verifiable problems

- Definition:

  We say that $L \subseteq \{0,1\}^*$ is **verifiable in time T** if there's a TM V st:

  - $x \in L \quad \Rightarrow \quad \exists\ w \in \{0,1\}^{\leq T(|x|)}, \qquad V(x,w) = 1$
  - $x \notin L \quad \Rightarrow \quad \forall\ w \in \{0,1\}^{\leq T(|x|)}, \qquad V(x,w) = 0$
  - V(x,w) runs in time O(T(|x|))

Efficiently verifiable problems

- Example 1:

  Deciding if an integer can be factored is verifiable in polynomial time.

- L = { n : n is a composite number }

Efficiently verifiable problems

- Example 2:

  Deciding if an n-vertex G has a clique of size n/2 is verifiable in polynomial time.

- L = { G : G is an n-vertex graph with a clique of size n/2 }

Efficiently verifiable problems

› Definition:

NTIME[T] = { L : L is verifiable in time T } // non-deterministic time

$\mathbf{NP} = U_c \text{ NTIME}[n^c]$

Efficiently verifiable problems

› Definition:

NTIME[T] = { L : L is verifiable in time T } // non-deterministic time

$\mathbf{NP} = \bigcup_c \text{NTIME}[n^c]$

› Sanity check:

$\mathbf{P} \subseteq \mathbf{NP}$

P vs NP

- Conjecture:

  $\mathbf{P} \neq \mathbf{NP}$

P vs NP

- Conjecture:

  $\mathbf{P} \neq \mathbf{NP}$

In words: there's a problem whose solutions can be easily verified, but it's hard to find a solution.

P vs NP

- Conjecture:

  $\mathbf{P} \neq \mathbf{NP}$

- Still open!

In words: there's a problem whose solutions can be easily verified, but it's hard to find a solution.

P vs NP

- Conjecture:

  $\mathbf{P} \neq \mathbf{NP}$

- Still open!

In words: there's a problem whose solutions can be easily verified, but it's hard to find a solution.

- Most important open problem in theoretical computer science.
- Tremendous intellectual efforts, rich dividends across CS & math.

P vs NP

- Reflections:
  - Human intuition suggests that $\mathbf{P} \neq \mathbf{NP}$
  - The alternative $\mathbf{P} = \mathbf{NP}$ would have staggering implications (good & bad)
  - First-order approximation to “some everyday problems are infeasible”
  - Prerequisite to good things, e.g. provably secure cryptography

Roadmap

1. An equivalent "$\mathbf{P} \neq \mathbf{NP}$ for search problems"
2. NP-completeness: evidence of hardness
3. Stronger versions of $\mathbf{P} \neq \mathbf{NP}$

A search formulation

- $\mathbf{P} \neq \mathbf{NP}$ is phrased in terms of decision problems merely for convenience and simplicity

- Goal (informal):

  $\mathbf{P} \neq \mathbf{NP}$ ⇔ there's a verifiable **search problem** that is hard

A search formulation

- Reminder:
  - a **relation** is a set $R \subseteq \{0,1\}^* \times \{0,1\}^*$
    - think of each $(x,y) \in R$ as an input x and a solution y
    - think of R as the set of valid input-solution pairs
  - the **search problem R:**

    Given x, find y st $(x,y) \in R$, or say “error” if no such y exists.

A search formulation

- For a relation R, let

$$L_R = \{ x : \exists\ y, (x,y) \in R \}$$

be the set of inputs with valid solutions.

- A relation R is **polytime verifiable** if $R \in \mathcal{P}$ and there is $c > 1$ st $(x,y) \in R \Rightarrow |y| \leq |x|^c$

A search formulation

- Theorem:

  1. $L \in \mathbf{NP} \Rightarrow \exists$ polytime verifiable R st $L = L_R$
  2. R is polytime verifiable $\Rightarrow L_R \in \mathbf{NP}$

- We call R from Item (1) a **witness-relation** for L
- Can L have several different witness-relations?

Search-to-decision reduction

- Theorem:

  The following two statements are equivalent:

  1. $\mathbf{P} = \mathbf{NP}$
  2. Every polytime verifiable R is solvable in polytime.

Search-to-decision reduction

- Theorem:

  The following two statements are equivalent:

  1. $\mathbf{P} = \mathbf{NP}$
  2. Every polytime verifiable R is solvable in polytime.

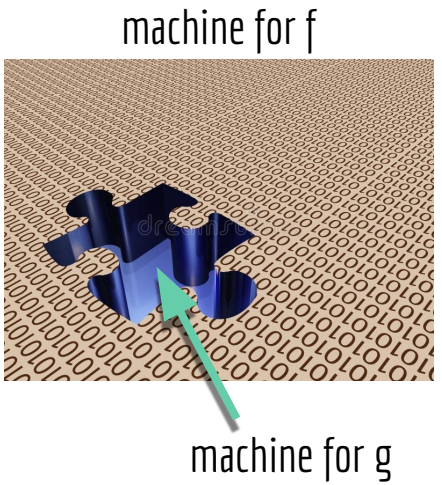
- Bottom-line:

  We study $\mathbf{P}$ vs $\mathbf{NP}$ using languages for convenience & simplicity

Reductions

- Recap:
  - if we have a machine for g, we can build a machine for f

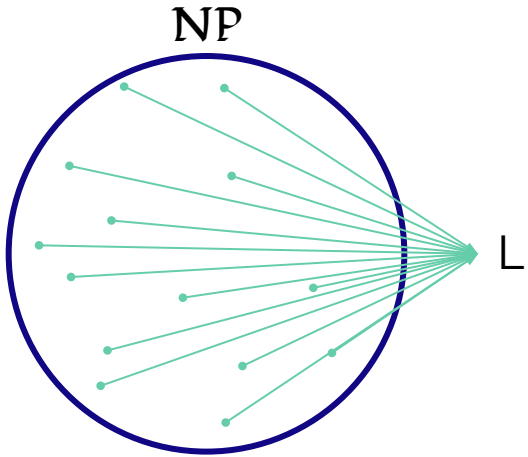
    " f reduces to g ", " f ≤ g "



NP-completeness

› Definition:

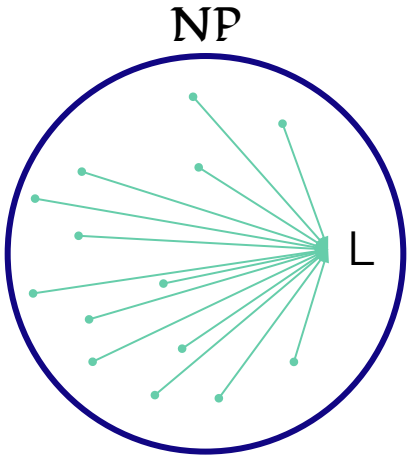
We say that $L \subseteq \{0,1\}^*$ is **NP-hard** if every problem in $\mathcal{NP}$ reduces to L in polynomial time.



NP-completeness

› Definition:

We say that $L \subseteq \{0,1\}^*$ is **NP-complete** if $L \in \mathcal{NP}$ and L is NP-hard.



NP-completeness

› Definition:

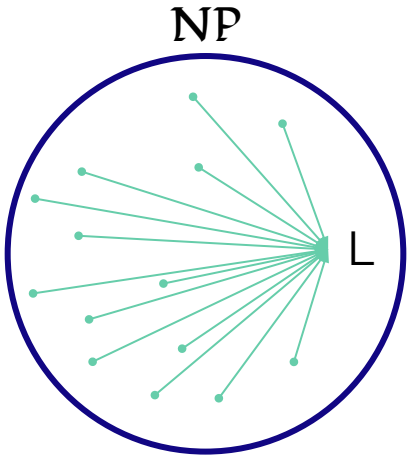
We say that $L \subseteq \{0,1\}^*$ is **NP-complete** if $L \in \mathcal{NP}$ and L is NP-hard.

› Meaning:

› intuitively: the “hardest” problems in $\mathcal{NP}$

› what happens if $L \in \mathcal{P}$?

› what happens if $\mathcal{P} \neq \mathcal{NP}$?



NP-completeness

- Theorem:

  There are $\mathbf{NP}$-complete problems.

NP-completeness

- Theorem:

  There are $\mathbf{NP}$-complete problems.

- Theorem (informal):

  There are **thousands** of **interesting** $\mathbf{NP}$-complete problems.

NP-completeness

- Theorem:

  There are $\mathbf{NP}$-complete problems.

- Theorem (informal):

  There are **thousands** of **interesting** $\mathbf{NP}$-complete problems.

  … including almost all of the $\mathbf{NP}$-problems mentioned so far

NP-completeness

- Theorem:

  There are $\mathbf{NP}$-complete problems.

- Theorem (informal):

  There are **thousands** of **interesting** $\mathbf{NP}$-complete problems.

  … including almost all of the $\mathbf{NP}$-problems mentioned so far

- Corollary: If $\mathbf{P} \neq \mathbf{NP}$, all of these problems are not in $\mathbf{P}$.

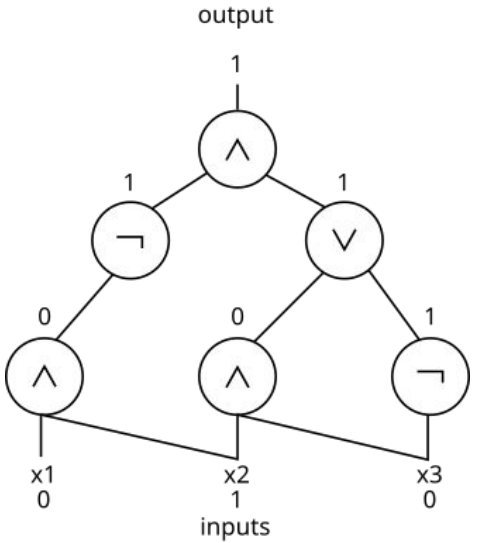
Efficiently computable problems

› Lower bounds: What lies outside $\mathbf{P}$?

NP-completeness

› Definition:

A **Boolean circuit** is

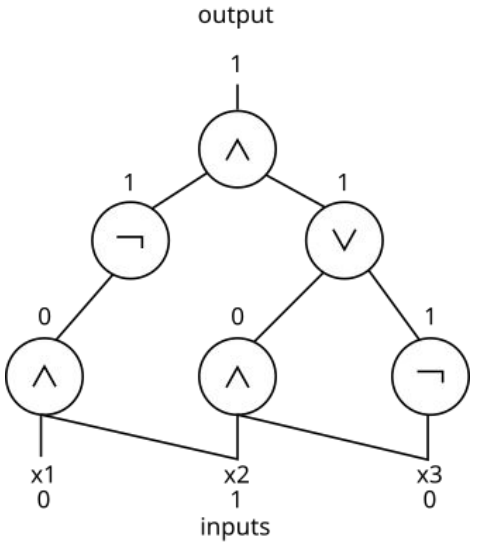


NP-completeness

- Definition:

  A **Boolean circuit** is a DAG with labeled nodes:

  - input: nodes labeled as $x_1,\ldots,x_n$, have fan-in 0
  - intermediary: nodes labeled with a function in $\{\wedge, \vee, \neg\}$, have fan-in 1 or 2
  - output: one node is labeled as “output”

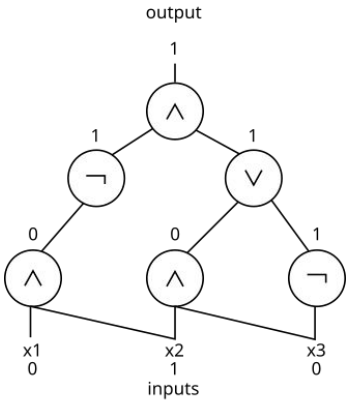


NP-completeness

› Definition:

A Boolean circuit C with n input nodes computes a function $C:\{0,1\}^n \to \{0,1\}$ in the obvious way.

(given bit-values for $x_1,\ldots,x_n$, propagate upwards)



NP-completeness

› Definition:

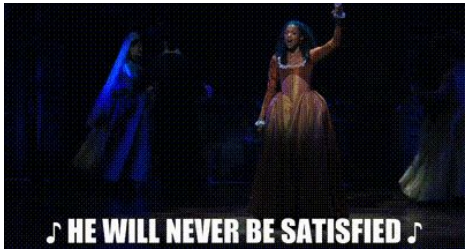
CircuitSAT = { <C> : C is a Boolean circuit and $\exists x,\ C(x) = 1$ }

NP-completeness

- Definition:

  CircuitSAT = { <C> : C is a Boolean circuit and $\exists x,\ C(x) = 1$ }

- Terminology:
  - x satisfies C if C(x) = 1, in which case x is a satisfying assignment
  - CircuitSAT is the satisfiability problem for Boolean circuits



NP-completeness

- Theorem:

  CircuitSAT is $\mathbf{NP}$-complete.

NP-completeness

- Definition:

  A **3-constraint** is

  - three indices: i, j, k
  - function: $f:\{0,1\}^3 \to \{0,1\}$

- We consider **lists** of 3-constraints
- An input x **satisfies the list** if every constraint in the list is satisfied

$x_1 = x_2 \wedge x_3$

$x_n = 1 \Rightarrow x_2 = x_7$

$x_7 \oplus x_3 \oplus x_{12} = 0$

$x_1 = x_2$ // note: 2 vars

$x_1 = 1$ // note: 1 var

~~$x_1 \vee x_2 \vee x_{n-1} \vee x_n = 1$~~

NP-completeness

› Definition:

3CSP = { <L> : L is a satisfiable list of 3-constraints }

NP-completeness

- Definition:

  3CSP = { <L> : L is a satisfiable list of 3-constraints }

- Theorem:

  3CSP is $\mathbf{NP}$-complete.

NP-completeness

- Definition:

  A **3CNF** is an AND of ORs where each OR is over three literals (= vars / negations of vars)

- Each OR_3 is called a “clause”
- An input x satisfies the 3CNF if each clause evaluates to 1 on x

$x_1 \vee \neg x_2 \vee x_3$

$x_n \vee x_2 \vee \neg x_7$

$\neg x_7 \vee \neg x_3 \neg x_{12}$

$x_1 \vee x_2 \vee \neg x_4$

…

…

NP-completeness

› Definition:

3SAT = { <C> : C is a satisfiable 3CNF }

NP-completeness

- Definition:

  3SAT = { <C> : C is a satisfiable 3CNF }

- Theorem:

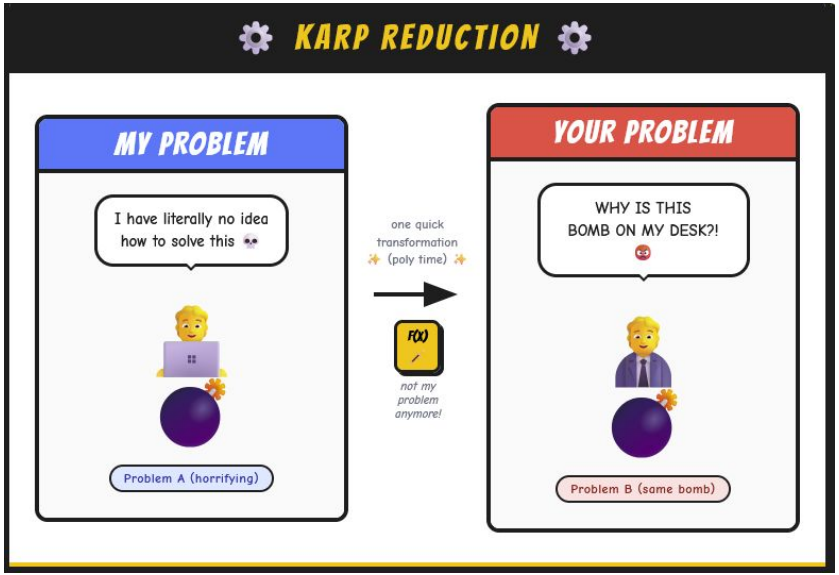
  3SAT is $\mathbf{NP}$-complete.

An aside

› Definition:

A **Karp reduction of L to L'** is a polynomial-time machine R st

$$x \in L \Leftrightarrow R(x) \in L'$$

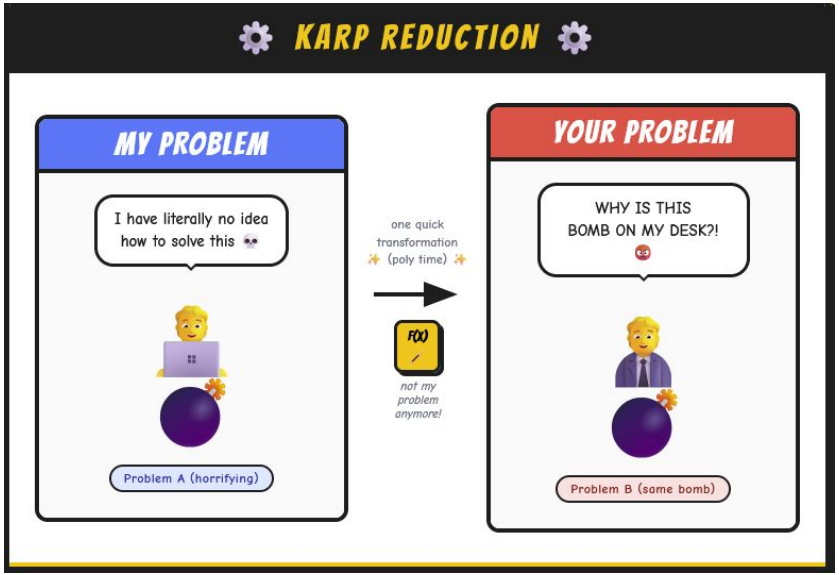


An aside

› Definition:

A **Karp reduction of L to L'** is a polynomial-time machine R st

$$x \in L \Leftrightarrow R(x) \in L'$$



› specific way of reducing L to L'

› we've been using it for a while

› common in $\mathbf{NP}$-completeness proofs (often it's enough)

Efficiently computable problems

› Lower bounds: What lies outside $\mathbb{P}$?

$\mathbb{NP}$-complete:

SAT variants:
CircuitSAT, 3CSP, 3SAT, …

NP-complete graph problems

- Definition:

  In the **independent set** problem we are given a graph G and an integer k and need to decide if G has an independent set of size k.

- As a language,

  IND = { (G,k) : G has an independent set of size k }

NP-complete graph problems

- For an n-vertex graph, input length is $O(n^2 + \log(k)) = O(n^2)$.
- Trivial algorithm runs in time $O(n^k)$, best known algorithms run in time $n^{\theta(k)}$ or $2^{\theta(n)}$.
  - when $k = \omega(1)$ super-polynomial time
  - when $k = \Omega(n)$ exponential time

NP-complete graph problems

- Theorem:

  IND is $\mathbf{NP}$-complete.

NP-complete graph problems

- Theorem:

  IND is $\mathbf{NP}$-complete.

- Corollary:

  CLIQUE is $\mathbf{NP}$-complete.

  CLIQUE = { (G,k) : G has a clique of size k }

NP-complete graph problems

- Definition:

  We say that a graph G=(V,E) is **3-colorable** if there's c: V → {1,2,3} such that for every (u,v) ∈ E it holds that c(u) ≠ c(v).

  In the **3-colorability** problem we are given a graph G and need to decide if G is 3-colorable. (define as a language in the obvious way.)

NP-complete graph problems

- For an n-vertex graph, the input length is n^2.
- Brute-force algorithm in time $3^n \cdot \text{poly}(n)$.
  - easy improvement to $2^n \cdot \text{poly}(n)$.
- Best known algorithm runs in time $2^{\theta(n)}$.

NP-complete graph problems

- Theorem:

  3COLOR is $\mathbf{NP}$-complete.

Efficiently computable problems

› Lower bounds: What lies outside $\mathbf{P}$?

$\mathbf{NP}$-complete:

SAT variants:
CircuitSAT, 3CSP, 3SAT, …

Graph problems:
IND, CLIQUE, 3COLOR, …

Algebraic problems

› Definition:

In **LIN** we are given a system of linear equations modulo 2, and we need to decide if there's a solution.

Algebraic problems

- Definition:

  In **LIN** we are given a system of linear equations modulo 2, and we need to decide if there's a solution.

- Theorem:

  LIN $\in \mathcal{P}$.

  (You know this! Just use Gaussian elimination.)

Algebraic problems

› Definition:

In **QUAD** we are given a system of quadratic equations modulo 2, and we need to decide if there's a solution.

Algebraic problems

- Definition:

  In **QUAD** we are given a system of quadratic equations modulo 2, and we need to decide if there's a solution.

- Theorem:

  QUAD is $\mathbf{NP}$-complete.

Efficiently computable problems

› Lower bounds: What lies outside $\mathbf{P}$?

$\mathbf{NP}$-complete:

SAT variants:
CircuitSAT, 3CSP, 3SAT, …

Graph problems:
IND, CLIQUE, 3COLOR, …

Algebra:
QUAD, …

NP-complete optimization problems

- Definition:

  In the **subset-sum** problem we are given $w_1,\ldots,w_n \in \mathbb{N}$ and $W \in \mathbb{N}$ and need to decide if there's a subset $S \subseteq [n]$ st $\sum_{s\in S} w_s = W$.

  (define as a language in the obvious way.)

NP-complete optimization problems

- Definition:

  In the **subset-sum** problem we are given $w_1,\ldots,w_n \in \mathbb{N}$ and $W \in \mathbb{N}$ and need to decide if there's a subset $S \subseteq [n]$ st $\sum_{s\in S} w_s = W$.

  (define as a language in the obvious way.)

- Theorem:

  SUBSET-SUM is $\mathbf{NP}$-complete.

NP-complete optimization problems

- Definition:

  In the **Knapsack** problem we get $v_1,\dots,v_n,w_1,\dots,w_n \in \mathbb{N}$ and $W, k \in \mathbb{N}$ and want to decide if there's $S \subseteq [n]$ st $\sum_{i\in S} v_i \geq k$ and $\sum_{i\in S} w_i \leq W$.

  (define as a language in the obvious way.)

NP-complete optimization problems

- Definition:

  In the **Knapsack** problem we get $v_1,\dots,v_n,w_1,\dots,w_n \in \mathbb{N}$ and $W, k \in \mathbb{N}$ and want to decide if there's $S \subseteq [n]$ st $\sum_{i\in S} v_i \geq k$ and $\sum_{i\in S} w_i \leq W$.

  (define as a language in the obvious way.)

- Reminder (from C73):

  There is an algorithm (dynamic programming) solving Knapsack in time $O(n \cdot W)$ in the RAM model $\Leftarrow$ "pseudopolynomial"

NP-complete optimization problems

› Corollary of SUBSET-SUM being $\mathbf{NP}$-complete:

Knapsack is $\mathbf{NP}$-complete.

Efficiently computable problems

› Lower bounds: What lies outside $\mathbf{P}$?

$\mathbf{NP}$-complete:

SAT variants:
CircuitSAT, 3CSP, 3SAT, …

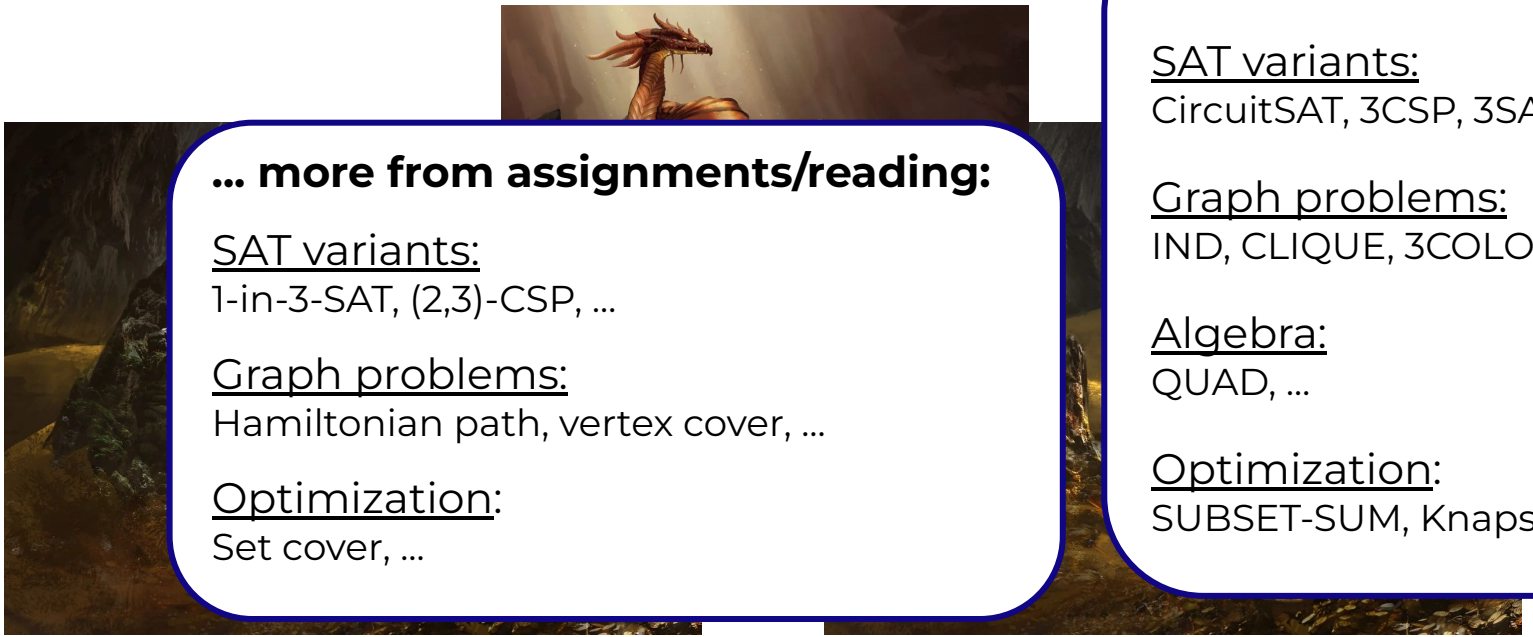
Graph problems:
IND, CLIQUE, 3COLOR, …

Algebra:
QUAD, …

Optimization:
SUBSET-SUM, Knapsack, …

Efficiently computable problems

› Lower bounds: What lies outside $\mathbf{P}$?



… more from assignments/reading:

SAT variants:
1-in-3-SAT, (2,3)-CSP, …

Graph problems:
Hamiltonian path, vertex cover, …

Optimization:
Set cover, …

$\mathbf{NP}$-complete:

SAT variants:
CircuitSAT, 3CSP, 3SAT, …

Graph problems:
IND, CLIQUE, 3COLOR, …

Algebra:
QUAD, …

Optimization:
SUBSET-SUM, Knapsack, …

NP-intermediate problems

- Weird:

  So far, all problems we saw are either in $\mathbf{P}$, or $\mathbf{NP}$-complete.

  (sometimes the threshold between the two was sharp, e.g. 2CSP vs 3CSP.)

  Is that a coincidence?

NP-intermediate problems

- Theorem: (yes, it's a coincidence!)

  If $\mathbf{P} \neq \mathbf{NP}$, then there are languages in $\mathbf{NP} \setminus \mathbf{P}$ that aren't $\mathbf{NP}$-hard.

NP-intermediate problems

› Theorem: (yes, it's a coincidence!)

If $\mathbf{P} \neq \mathbf{NP}$, then there are languages in $\mathbf{NP} \setminus \mathbf{P}$ that aren't $\mathbf{NP}$-hard.

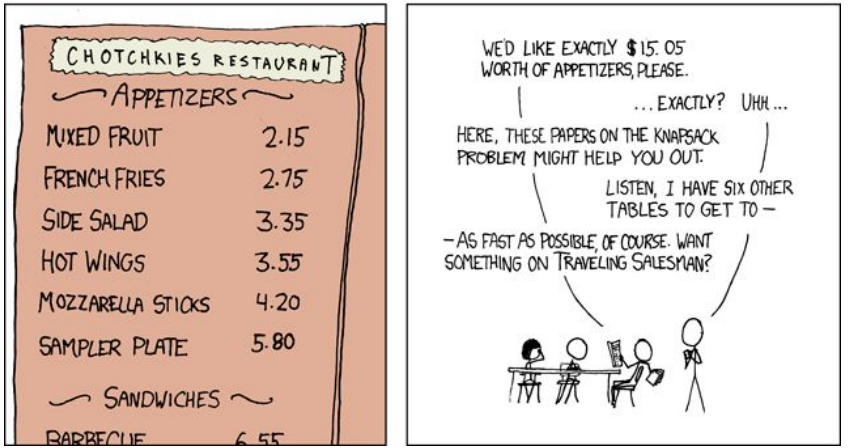
› Natural candidate (factoring): [1]

Given integers (N,k) in binary, is there a factor of N of size ≤ k?

(conjectured hardness of this problem is used heavily in cryptography.)

[1] there's no direct theoretical evidence for hardness, i.e. this candidate doesn't come from the theorem above

Final comments



source: https://xkcd.com/287/

Final comments

- Thousands of interesting problems are $\mathbf{NP}$-complete
  - we could spend an entire course showing more
  - lists of $\mathbf{NP}$-complete problems available in the textbooks and online
- Status of some interesting problems in $\mathbf{NP}$ is still undetermined
  - e.g., finding a minimal Boolean circuit computing a given function

Final comments

- Karp-reductions of the type we saw are often informally called "gadget reductions"
  - using simple combinatorial gadgets to map x to R(x)
- There are problems whose $\mathbf{NP}$-completeness relies on more complicated reductions
  - not all of them are Karp-reductions

Final comments

- Did we show any unconditional lower bound in this part of the course?

Final comments

- Did we show any unconditional lower bound in this part of the course?
- $\mathbf{NP}$-completeness gives conditional lower bounds:

  If $\mathbf{P} \neq \mathbf{NP}$, then $\mathbf{NP}$-complete problems can't be solved efficiently (i.e., in polynomial time)