6.890	Lecture 1	Sept. 4, 2014

6.890: Algorithmic Lower Bounds / Fun with Hardness Proofs

"Hardness made Easy"

Prof. Erik Demaine

TAs: Sarah Eisenstat & Jayson Lynch

http://courses.csail.mit.edu/6.890/fall14/

What is this class?

- practical guide to proving computational problems are formally hard / intractable
- NOT a complexity course
  (but we will use/refer to needed results)
- (anti)algorithmic perspective

Why take this class?

- know your limits in algorithmic design
- master techniques for proving hardness
  - key problems
  - proof styles
  - gadgets
- cool connections between problems
- fun problems like Mario & Tetris
  (serious problems too)
- solve puzzles → publishable papers

Background: algorithms, asymptotics, combinatorics
- no complexity background needed
  (but also little overlap with a complexity class)

Requirements:

- fill out form (& join mailing list)
- attend lectures
- scribing 1-2 lectures
- ≈5 psets, 2-3 weeks each
- project & presentation
  - theory (attempt to solve/pose open problem)
  - survey (something not covered in class)
  - implement/visualize
  - Wikipedia
  - art (sculpture, etc. related to hardness)

Topics: (on handout/webpage)

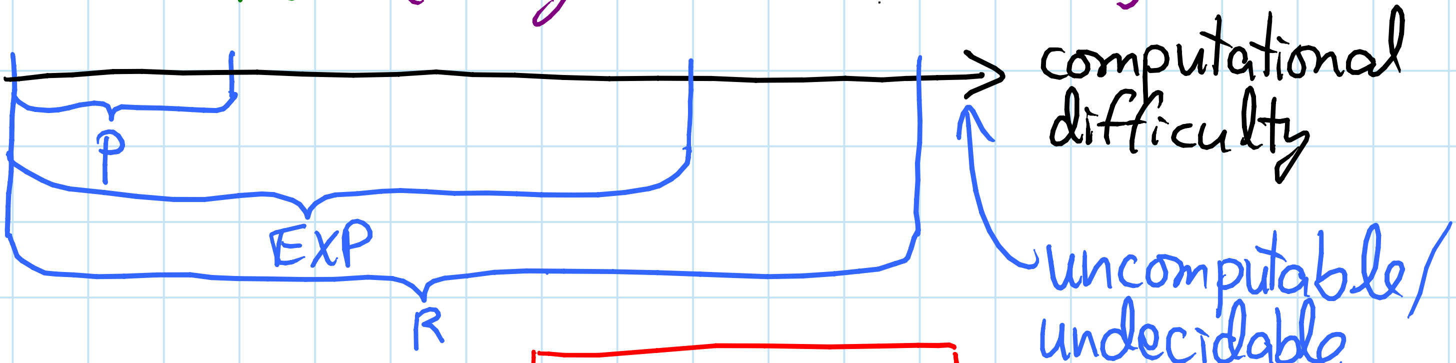
- NP-completeness (3SAT, 3-partition, Hamiltonicity, geometry)
- PSPACE, EXPTIME, ...
- Games, Puzzles, & Computation (Constraint Logic, Sudoko, Nintendo, Tetris, Rush Hour, Chess, Go, ...)
- inapproximability (PCP, APX, Set Cover, ind. set, UGC, ...)
- fixed-parameter intractability (W, clique, ...)
- 3SUM (toward n^2)
- counting (#P) & uniqueness (ASP)
- economic game theory (PPAD)
- existential theory of reals, undecidability (if time)

Recommended texts: Games, Puzzles, & Computation [Hearn & Demaine]
Garey & Johnson

Intro to complexity:

P = {problems solvable in polynomial time} ($\to n^c$)
EXP = {problems solvable in exponential time} ($\to 2^{n^c}$)

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



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

Examples:

- negative-weight cycle detection $\in P$
- $n \times n$ Chess $\in EXP$ but $\notin P$
  ↳ who wins from given board config.?
- Tetris $\in EXP$ but don't know whether $\in P$
  ↳ survive given pieces from given board
- halting problem $\notin R$
- "most" decision problems $\notin R$
  (# algorithms $\approx \mathbb{N}$; # dec. problems $\approx 2^{\mathbb{N}} = \mathbb{R}$)

answer ∈ {YES, NO}

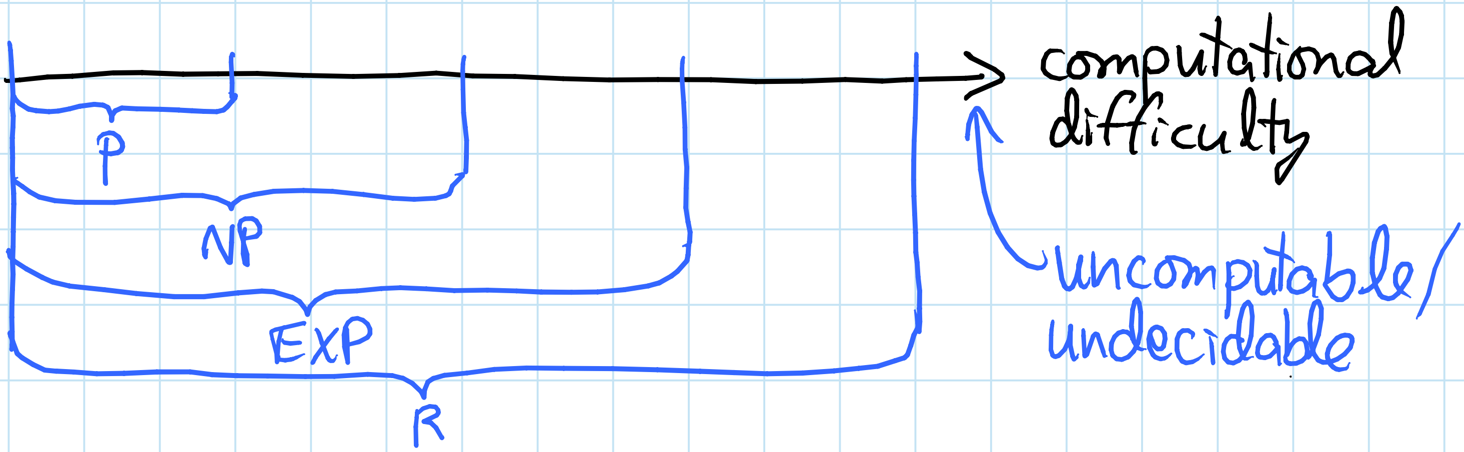
<u>NP</u> = {decision problems solvable in poly. time via a "<u>lucky</u>" algorithm}

can make lucky guesses, always "right", <u>without</u> trying all options

- <u>nondeterministic model</u>: algorithm makes guesses & then says YES or NO
- guesses guaranteed to lead to YES outcome if possible (NO otherwise)

= {decision problems with solutions that can be "<u>checked</u>" in polynomial time}

- when answer = YES, can "<u>prove</u>" it & poly.-time algorithm can check proof



<u>Example</u>: Tetris ∈ NP

- nondeterministic alg:
  - guess each move
  - did I survive?
- proof of YES: list what moves to make (<u>rules</u> of Tetris are easy)

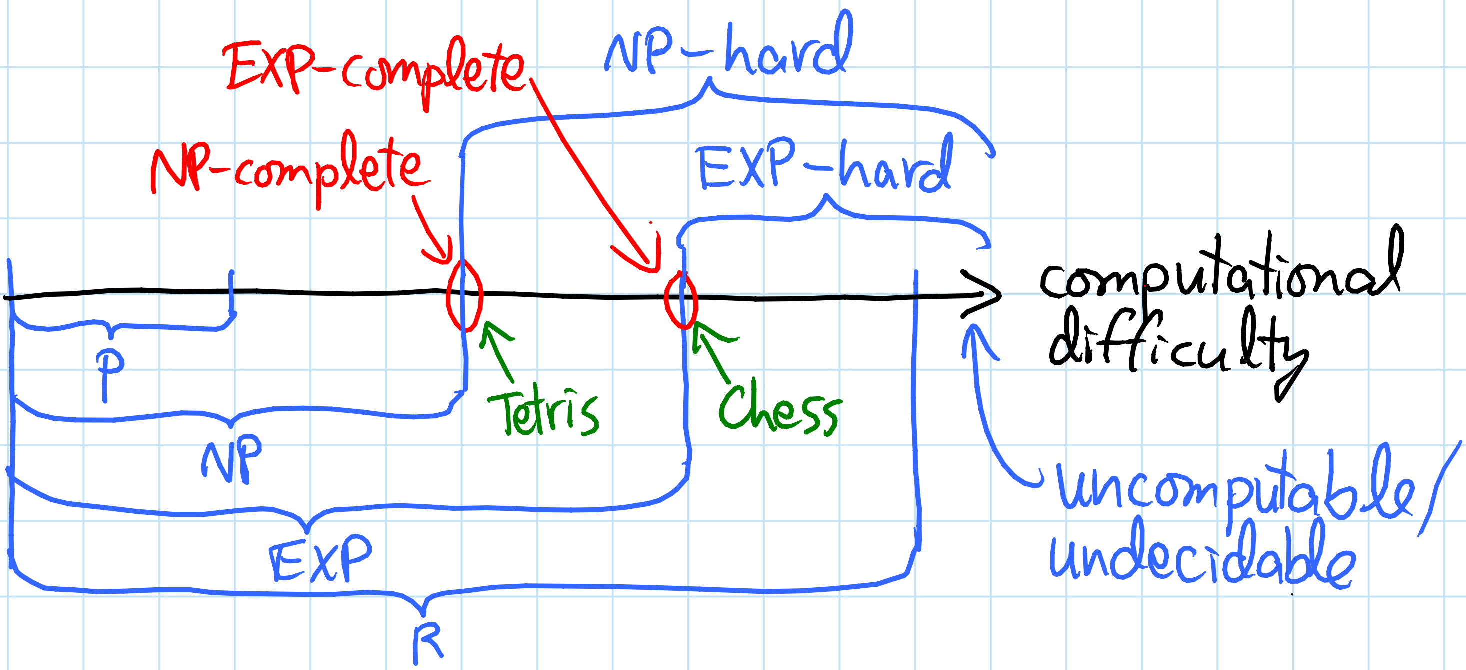
P ≠ NP: big conjecture (worth \$1,000,000)
≈ can't engineer luck
≈ generating (proofs of) solutions can be harder than checking them

CoNP = negations (YES ⇔ NO) of problems ∈ NP
= problems with good proofs of NO answer

NP, EXP, etc.
defined later

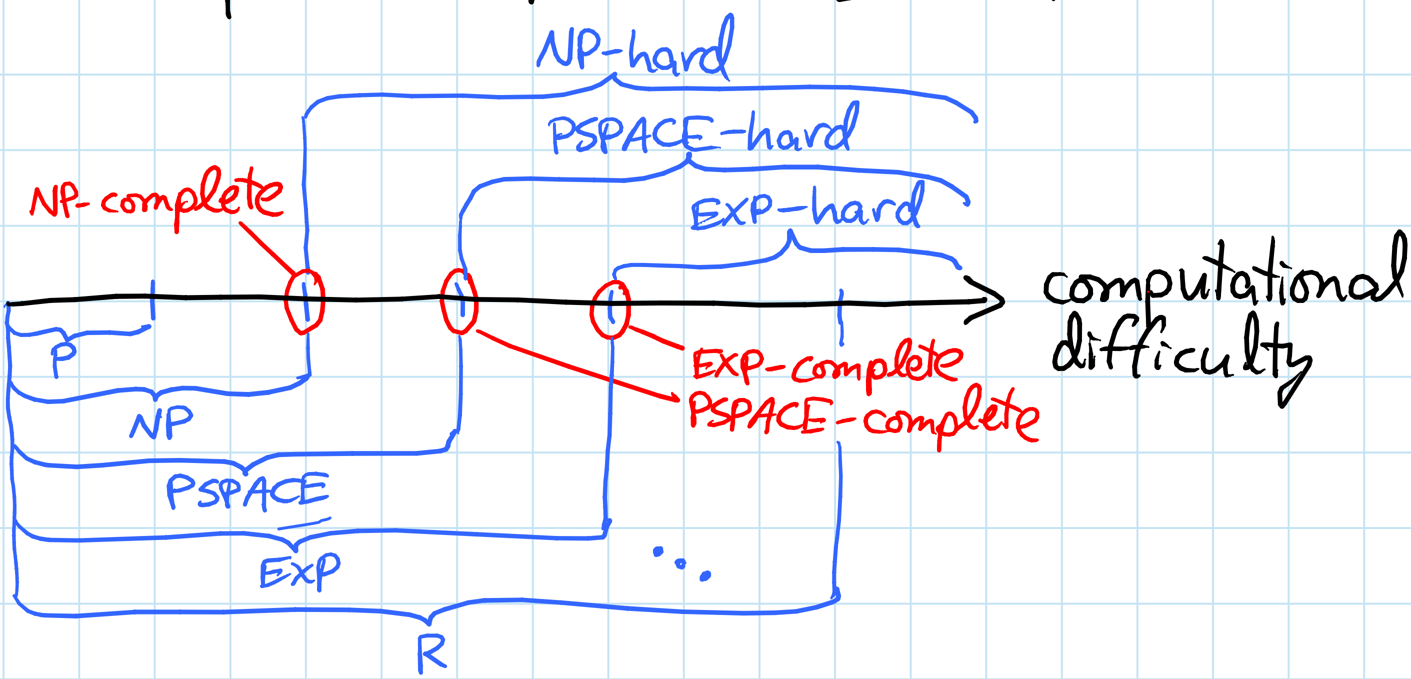
X-hard = "as hard as" every problem ∈ X
X-complete = X-hard ∩ X
sometimes "X-easy" = ∈ X

e.g. Tetris is NP-complete
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2004]
⇒ if P ≠ NP, then Tetris ∈ NP ∖ P



e.g. Chess is EXP-complete ⇒ ∉ P
⇒ Chess ∈ EXP ∖ NP if NP ≠ EXP (also open)

PSPACE = {problems solvable in polynomial space}
- $\subseteq$ EXP: only exponentially many states
- $\supseteq$ NP: simulate all executions, take running OR
- open whether either is strict



e.g. Rush Hour is PSPACE-complete [Flake & Baum 2002]
$\Rightarrow \notin$ P if P $\neq$ NP <u>or</u> NP $\neq$ PSPACE

<u>Beyond exponential:</u> (not too important)

EXP(TIME) $\subseteq$ EXPSPACE $\subseteq$ 2EXP(TIME) $\subseteq$ 2EXPSPACE $\subseteq \cdots$

double exponential: $2^{2^{n^c}}$

Also L = LOGSPACE → $O(\lg n)$ bits of space!

EXP $\subsetneq$ 2EXP $\subsetneq \cdots$ ← time & space hierarchy theorems

L $\subsetneq$ PSPACE $\subsetneq$ EXPSPACE $\subsetneq$ 2EXPSPACE $\subsetneq \cdots$

<u>Nondeterministic:</u>

in general, space bound squares
- NPSPACE = PSPACE [Savitch 1970] (very useful!)
- NEXP, N2EXP, ...: analogs of NP

What does "as hard as" mean?

(other constraints possible)

Reduction from A to 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

$B \in P$ ⋮ $A \in P$
$B \in NP$ ⋮ $A \in NP$

⇒ B is at least as hard as A
(A is a special case of B)

- 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

Examples from algorithms:

- unweighted shortest paths → weighted $(w=1)$
- min-product path → min-sum path $(\lg)$
- longest path → shortest path (negate)
- min-weight k-step path → min-weight path
  (k copies of graph + links between adj. layers)

Almost all hardness proofs are by reduction <u>from</u> known hard problem <u>to</u> your problem

Examples of hardness proofs:

① Super Mario Bros. is NP-complete

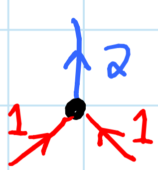
- reduction from 3SAT: can you satisfy (make true) a formula like

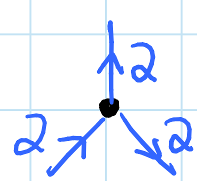
$$(x_1 \text{ OR } x_2 \text{ OR } x_3) \quad \text{variable}$$
$$\text{AND } (x_5 \text{ OR } \underbrace{(\text{NOT } x_3)}_{\text{literal}} \text{ OR } \underbrace{x_4}_{\text{literal}}) \ \cdots$$
$$\text{AND } \cdots$$

3 literals per clause

② Rush Hour is PSPACE-complete

- reduction from NCL (Nondet. Constraint Logic): given directed graph with edge weights $\in \{1, 2\}$, find sequence of edge reversals to reverse a target edge, while at all times maintaining total in-weight ≥ 2 at each vertex
- only need AND vertex & OR vertex





(in fact: OR can be protected: only one input active at once)

- Constraint Logic is a powerful tool for proving hardness of games & puzzles

6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs
Fall 2014