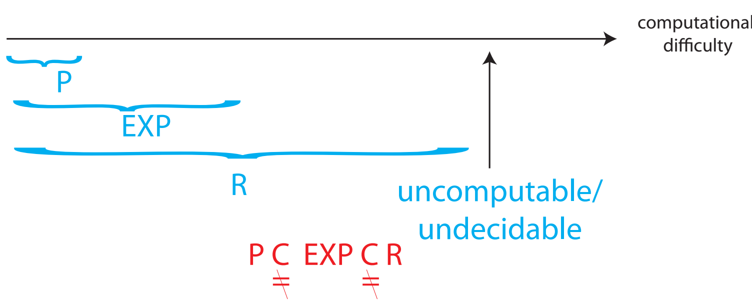
Lecture 23: Computational Complexity

Lecture Overview

- P, EXP, R
- Most problems are uncomputable
- NP
- Hardness & completeness
- Reductions

Definitions:

- P = {problems solvable in polynomial (n^c) time}
  (what this class is all about)
- EXP = {problems solvable in exponential (2^{n^c}) time}
- R = {problems solvable in finite time} "recursive" [Turing 1936; Church 1941]



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 don't know whether $\in$ P
  Survive given pieces from given board.

Halting Problem:

Given a computer program, does it ever halt (stop)?

- uncomputable ($\notin$ R): no algorithm solves it (correctly in finite time on all inputs)
- decision problem: answer is YES or NO

Most Decision Problems are Uncomputable

- program $\approx$ binary string $\approx$ nonneg. integer $\in N$
- decision problem = a function from binary strings ($\approx$ nonneg. integers) to {YES (1), NO (0)}
- $\approx$ infinite sequence of bits $\approx$ real number $\in \mathbb{R}$
  $|\mathbb{N}| \ll |\mathbb{R}|$: no assignment of unique nonneg. integers to real numbers ($\mathbb{R}$ uncountable)
- $\Longrightarrow$ not nearly enough programs for all problems
- each program solves only one problem
- $\Longrightarrow$ almost all problems cannot be solved

NP

NP = {Decision problems solvable in polynomial time via a "lucky" algorithm}. The "lucky" algorithm can make lucky guesses, always "right" without trying all options.

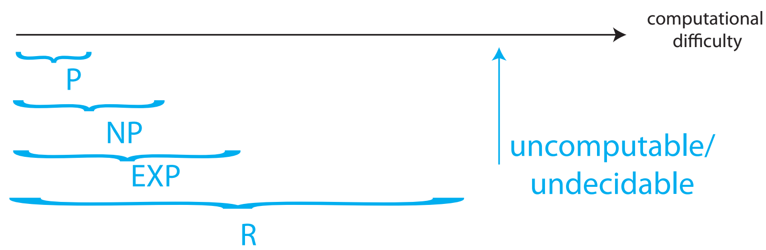
- nondeterministic model: algorithm makes guesses & then says YES or NO
- guesses guaranteed to lead to YES outcome if possible (no otherwise)

In other words, NP = {decision problems with solutions that can be "checked" in polynomial time}. This means that when answer = YES, can "prove" it & polynomial-time algorithm can check proof

Example

Tetris $\in$ NP

- nondeterministic algorithm: guess each move, did I survive?
- proof of YES: list what moves to make (rules of Tetris are easy)



P $\neq$ NP

Big conjecture (worth $1,000,000)

- $\approx$ cannot engineer luck
- $\approx$ generating (proofs of) solutions can be harder than checking them

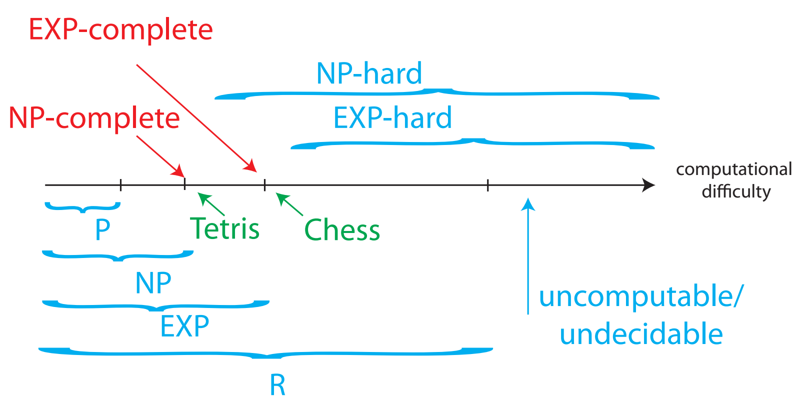
Hardness and Completeness

Claim:

If P $\neq$ NP, then Tetris $\in$ NP - P
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2004]

Why:

Tetris is NP-hard = "as hard as" every problem $\in$ NP. In fact NP-complete = NP $\cap$ NP-hard.



Similarly

Chess is EXP-complete = EXP $\cap$ EXP-hard. EXP-hard is as hard as every problem in EXP. If NP $\neq$ EXP, then Chess $\notin$ EXP $\setminus$ NP. Whether NP $\neq$ EXP is also an open problem but less famous/"important".

Reductions

Convert your problem into a problem you already know how to solve (instead of solving from scratch)

- most common algorithm design technique
- unweighted shortest path $\rightarrow$ weighted (set weights = 1)
- min-product path $\rightarrow$ shortest path (take logs) [PS6-1]
- longest path $\rightarrow$ shortest path (negate weights) [Quiz 2, P1k]
- shortest ordered tour $\rightarrow$ shortest path (k copies of the graph) [Quiz 2, P5]
- cheapest leaky-tank path $\rightarrow$ shortest path (graph reduction) [Quiz 2, P6]

All the above are One-call reductions: A problem $\rightarrow$ B problem $\rightarrow$ B solution $\rightarrow$ A solution
Multicall reductions: solve A using free calls to B — in this sense, every algorithm reduces problem $\rightarrow$ model of computation
NP-complete problems are all interreducible using polynomial-time reductions (same difficulty). This implies that we can use reductions to prove NP-hardness — such as in 3-Partition $\rightarrow$ Tetris

Examples of NP-Complete Problems

- Knapsack (pseudopoly, not poly)
- 3-Partition: given n integers, can you divide them into triples of equal sum?
- Traveling Salesman Problem: shortest path that visits all vertices of a given graph — decision version: is minimum weight $\leq x$?
- longest common subsequence of k strings
- Minesweeper, Sudoku, and most puzzles
- SAT: given a Boolean formula (and, or, not), is it ever true? x and not $x \rightarrow$ NO
- shortest paths amidst obstacles in 3D

- 3-coloring a given graph
- find largest clique in a given graph

6.006 Introduction to Algorithms
Fall 2011