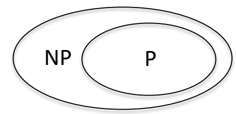
NP Hardness/Completeness Overview

Ron Parr
CompSci 570
Duke University
Department of Computer Science

Why Study NP-hardness

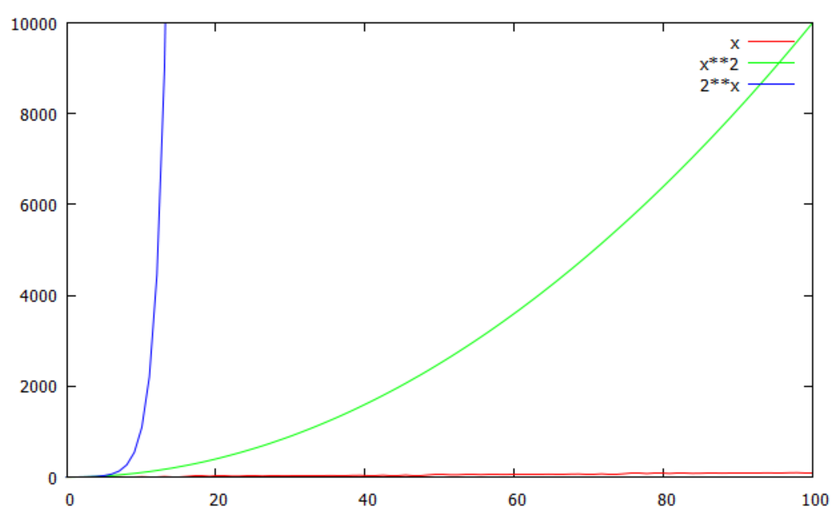
- NP hardness is not an AI topic, but…
- It's important for all computer scientists
- It has a particularly profound impact on AI because so many AI problems are NP-hard, e.g.,
  - Finding global minimum in neural network training
  - Planning and scheduling
  - Computing probabilities in Bayesian networks
  - Constraint satisfaction
  - Finding an equilibrium in a game that guarantees a minimum utility
  - Etc.

P and NP



- P and NP are about decision problems
- P is set of problems that can be solved in polynomial time
- NP is a (proper?) superset of P
- NP is the set of problems that:
  - Have solutions which can be verified in polynomial time or, equivalently,
  - can be solved by a non-deterministic Turing machine in polynomial time (a non-deterministic Turing machine can be thought of as a guess and check algorithm that always happens to guess correctly)
- Roughly speaking:
  - Problems in P are tractable – can be solved in a reasonable amount of time, and faster computers help
  - Some problems in NP *might* not be tractable – unknown if a poly time solution exists

Scaling – Why P matters

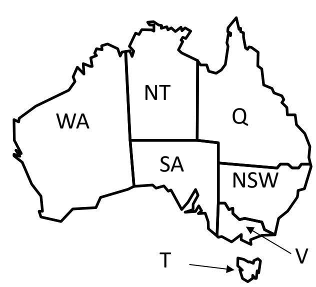


Isn't P big?

- P includes O(n), $O(n^2)$, $O(n^{10})$, $O(n^{100})$, etc.
- Clearly $O(n^{10})$ isn't something to be excited about – not practical
- Computer scientists are very clever at making things that are in P efficient
- First algorithms for some problems are often quite expensive, e.g., $O(n^3)$, but research often brings this down

Better understanding the class NP

- A class of *decision problems* (Yes/No)
- Solutions can be verified in polynomial time
- Examples:
  - Graph coloring:



  - Sortedness: [1 2 3 4 5 8 7]

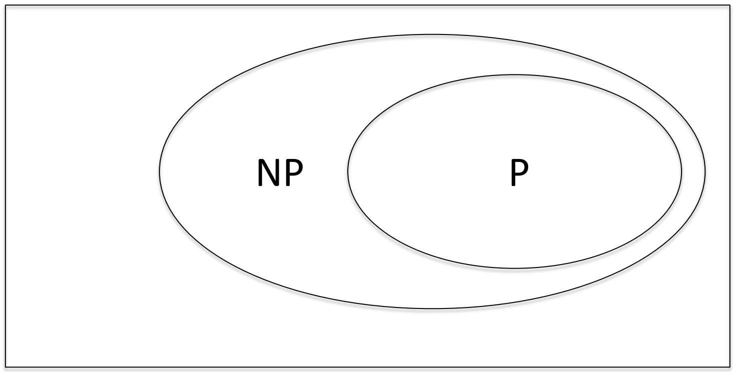
NP-hardness

- Many problems in AI are NP-hard (or worse)
- What does this mean?
- These are some of the hardest problems in CS
- Identifying a problem as NP hard means:
  - You probably shouldn't waste time trying to find a polynomial time solution
  - If you find a polynomial time solution, either
    - You have a bug
    - Find a place on your shelf for your Turing award
- NP hardness is a major triumph (and failure) for computer science theory

Hardness vs. Completeness

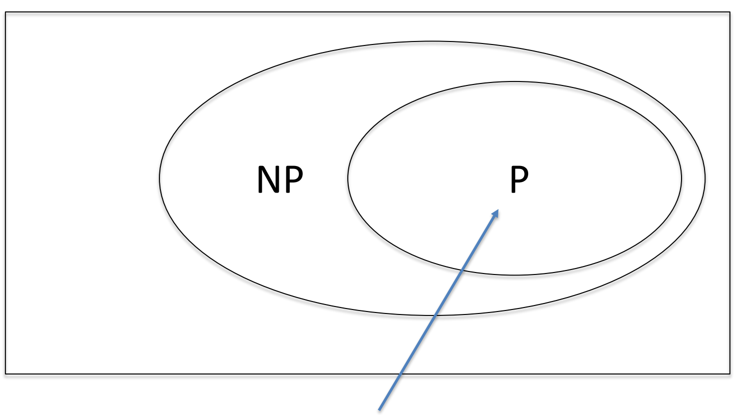
- If something is hard for a class (e.g. NP-hard) it is at least as hard as the hardest problems in class.
- If something is complete for a class (e.g. NP-complete) it must be hard and in the class.
- If something is NP-*hard*, it ***could be even harder*** than the hardest problems in NP

Hardness vs. Completeness Examples and Pictures



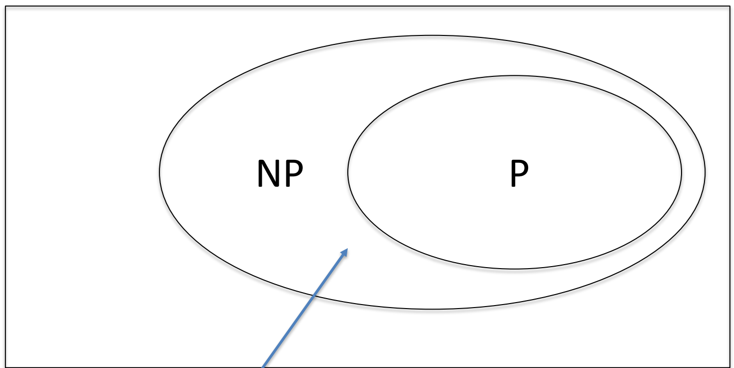
P=NP?
- Is P a proper subset of NP?
- Biggest open question in theoretical computer science

Hardness vs. Completeness Examples and Pictures



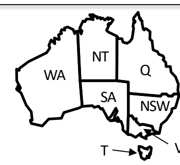
Consider an O(nlog(n)) problem, e.g., “Is this list sorted?”:
- In P
- In NP
- Can’t be NP-complete or NP-hard

Hardness vs. Completeness Examples and Pictures



Consider a graph coloring problem:

- Known to be NP-hard
- In NP
- Is NP-complete and NP-hard
- In P only if P=NP



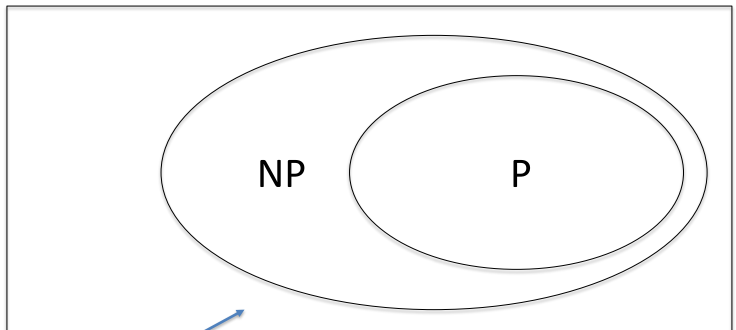
What’s harder still?

- PSPACE hardness
- Algorithms in P-space require polynomial space

- Why is this at least as hard as P-time?
- Example problem(s):
  - Some planning problems
  - Super Mario Bros. (not any actual game level) is PSPACE-hard

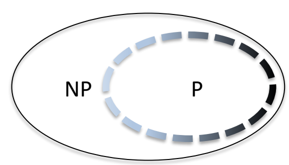
- PSPACE is a (proper?) superset of NP
- Still harder: exp-time

Hardness vs. Completeness Examples and Pictures



Consider Super Mario Brothers:
- PSPACE-Hard
- PSPACE-Hardness implies NP-Hardness
- In P? (only if P=NP)
- In NP? (only if NP=PSPACE)
- NP-Complete (only if NP=PSPACE)

P=NP?



- Biggest open question in CS
- Can NP-complete problems be solved in poly time?
- ***Probably not***, but nobody has proved it yet
- Somewhat recent attempt at proof detailed in NY Times, one of many false starts: http://www.nytimes.com/2009/10/08/science/Wpolynom.html

How challenging is "P=NP?"

- Princeton University CS department
- See: http://www.cs.princeton.edu/general/bricks.php
- Photo from: http://stuckinthebubble.blogspot.com/2009/07/three-interesting-points-on-princeton.html

NP-hardness

- Why it is a failure:
  - Huge class of problems w/o efficient solutions
  - We have failed, as a community, find efficient solutions *or prove that none exist*
- Why it is a triumph:
  - Precise language for talking about these problems
  - Sophisticated ways to reason about and categorize the problems we don't know how to solve efficiently
  - Developing an arsenal of approximation algorithms

Generic Examples of NP-Complete Problems

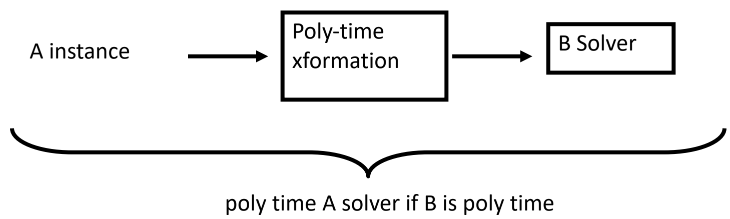
- ≥ 3 coloring
- Clique
- Set cover & vertex cover
- Traveling salesman
- Knapsack
- Subset sum
- Many, many, more…

How this impacts ***YOU***

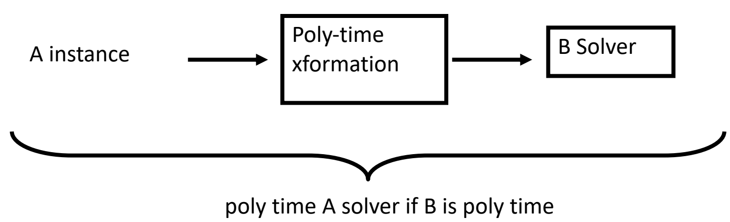
- *Not just a theoretical exercise*
- When confronted with a new problem:
  - Is this problem in P?
    (Confirm by finding a poly time algorithm)
  - Can't find a poly time algorithm?
    - Invest more effort?
    - Try to prove the problem is NP-hard?
  - If NP-hard, try to find effective heuristics that work in common cases, or try to find an effective approximation algorithm

Navigating the class NP

- An NP hard problem is at least has hard as the hardest problems in NP
- The hardest problems in NP are *NP-complete* (no known poly time solution)
- Demonstrate hardness via *reduction*
  - Use one problem to solve another
  - A is reduced to B, if we can use B to solve A:



Reductions



- Q: If B is NP-hard and A is of unknown difficulty, what does this tell us?
- Q: If A is NP-hard, and B is of unknown difficulty, what does this tell us?

SAT-The First NP-Complete Problem

- Given a set of binary variables
- Conjunction of disjunctions of these variables

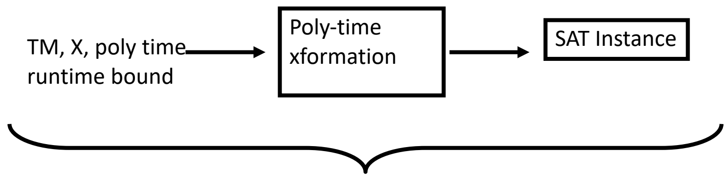
$$(x_1 \vee \overline{x_3} \vee x_7) \wedge (\overline{x_1} \vee x_{12} \vee x_9) \wedge \cdots$$

- Does there exist a satisfying assignment? (assignment that makes the expression evaluate to true)

How To Prove SAT is NP-Complete?

- Note: Clearly in NP
- Challenge: Nothing from which to reduce because this was the first NP-complete problem
- Idea (Cook 1971):
  - Input:
    - Any non-deterministic Turing machine - TM
    - Any input to that Turing machine - X
    - A polynomial bound on the run time of the machine
  - Output: A polynomial size SAT expression which evaluates to true IFF TM says YES – i.e., is there a path through tree of possible computations that evaluates to YES
- Conclusion: Solving SAT in poly time implies solving any problem in NP in poly time

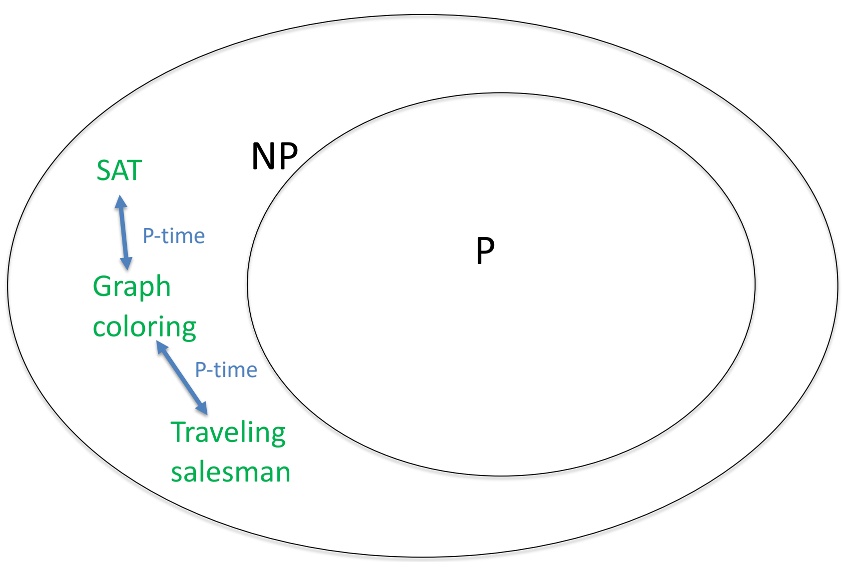
Cook's Result in a Cartoon



Assumptions: TM is a non-deterministic Turing machine with polynomial run time, i.e., a solver for problems in NP.

Poly time solver for SAT would solve any problem in NP in Poly time

Why NP-*completeness* is SO important



Why NP-*completeness* is SO important

- All NP-complete problems:
  - Are in NP
  - Got there by poly time transformation
  - Can solve any other problem in NP after poly time transformation
- Solving any one NP-complete problem in poly time unlocks ALL NP-complete problems!
- Cracking just one means P=NP!

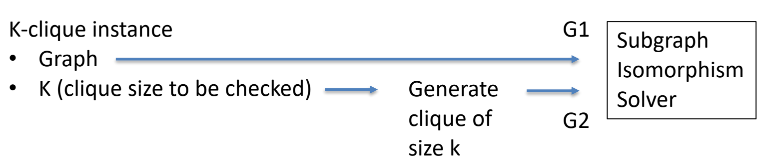
Easiest Hardness Proof: Proving Hardness Through Generalization

- Show problem A is NP-hard because known NP-hard problem B is a special case of A
- Example – generalizations of 3-SAT:
  - KSAT (k variables/clause) is NP-hard for any k>=3
  - SAT (no restrictions) generalizes 3SAT
  - Every valid 3SAT instance is a valid (K)SAT instance
  - A poly-time (K)SAT solver would ALSO be a poly time 3SAT solver
  - Conclusion: (K)SAT is at least as hard as 3SAT: NP-hard
- Trivial example of a reduction (transformation is a no-op)

k-clique -> Subgraph Isomorphism

- k-clique: Given G=(V,E), there exist a fully connected component of size k?
- Subgraph isomorphism: Given graphs G and H, does there exist a subgraph of G that is isomorphic to H
- (isomorphic = identical up to node relabelings)

Reduction



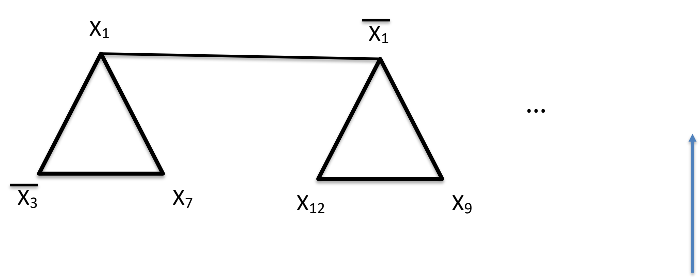
(Almost as simple a generalization)

Reduction: 3SAT -> Ind. Set

- Independent set: Given G=(V,E), does there exist a set of vertices of size k such that no two share an edge?
- Reduce 3SAT to independent set:
  - 3 nodes for each clause (corresponding to variable settings), and connect them in a 3-clique
  - Connect all nodes with complementary settings of the same variable
  - Pick k = # of clauses

Reduction Visualized

$$(x_1 \vee \overline{x_3} \vee x_7) \wedge (\overline{x_1} \vee x_{12} \vee x_9) \wedge \cdots$$



Keep adding one triangle for each clause

Optimization vs. Decision

- Optimization: Find the largest clique
- Decision: Does there exist a clique of size k

- NP is a family of ***decision*** problems
- In many cases, we can
  reduce decision to optimization
- (Use find-largest-clique to solve k-clique)

Weak vs. Strong Hardness

- Some problems can be brute-forced if the range of numbers involved is not large
- Subset sum: ∃ subset of a group of natural numbers that sums to k?
  - Suppose n numbers, largest of which is m
  - Initialize table of size k
  - Use dynamic programming
    - Iteration 1: Does there exist a set of size 1 that achieves each 1…k
    - Iteration j: Use iteration j-1 table to answer – Does there exist a set of size j that achieves each 1…k
    - Quit when j=n, or if you find a solution first
  - $O(kn^2)$ – polynomial in input size if magnitude of k is polynomial in input size (note: numbers are stored in binary!)
- Such problems are ***weakly NP-hard***

How To Avoid Embarrassing Yourself

- Don't say: "I proved that it requires exponential time." if you really meant:
  - "I proved it's NP-Hard/Complete"
  - "The best solution I could come up with takes exponential time."
- Don't say: "The problem is NP" (which doesn't even make sense) if you really meant:
  - "Problem is in NP" (often a weak statement)
  - "The problem NP-Hard/Complete" (usually a strong statement)
- Don't reduce new problems to NP-hard complete problems if you meant to prove the new problem is hard
- Such a reduction is backwards. What you really proved is that you can use a hard problem to solve an easy one. Always think carefully about the direction of your reductions

NP-Completeness Summary

- NP-completeness tells us that a problem belongs to class of similar, hard problems.

- What if you find that a problem is NP hard?
  - Look for good approximations with provable guarantees
  - Find different measures of complexity
  - Look for tractable subclasses
  - Use heuristics – try to do well on "most" cases