# NP Hardness/Completeness Overview

Ron Parr

CompSci 570

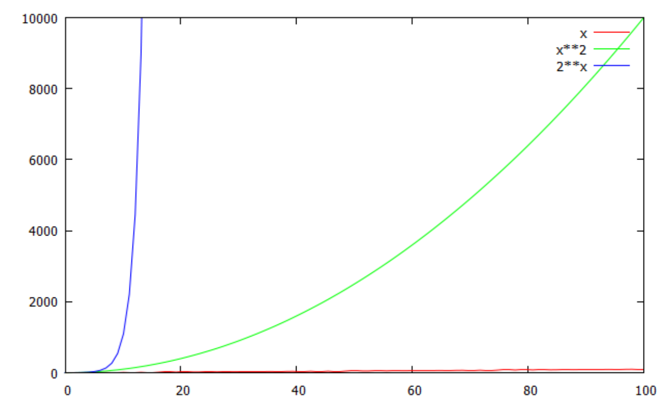
## Why Study NP-hardness

- NP hardness is not an AI topic
- It's important for all computer scientists
- Understanding it will deepen your understanding of AI (and other CS) topics
- You will be expected to understand its relevance and use for AI problems
- Eat your vegetables; they're good for you

## 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 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
- 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

## Scaling



## 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

## 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

## NP-hardness

- Why it is a failure:
  - Huge class of problems with no known efficient solutions
  - We have failed, as a community, find efficient solutions or prove that none exist
- Why it is a triumph:
  - Developed a precise language for talking about these problems
  - Developed sophisticated ways to reason about and categorize the problems we don't know how to solve efficiently
  - Developing an arsenal of approximation algorithms for hard problems

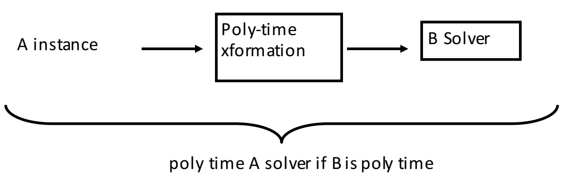
## Understanding the class NP

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

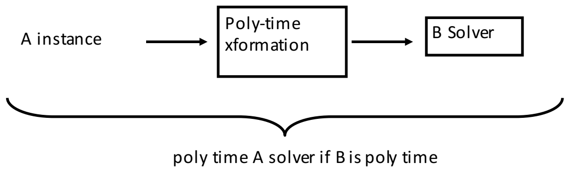

  - Sortedness: [1 2 3 4 5 8 7]

## What is NP hardness?

- 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




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

## Hardness vs. Completeness

- For something to be NP-*complete*, must be NP-hard and in NP
- If something is NP-*hard*, it ***could be even harder*** than the hardest problems in NP
- Proving completeness is stronger theoretical result – says more about the problem

## Examples of NP-Complete Problems

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

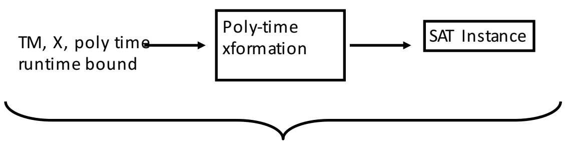
## SAT-The First NP-Complete Problem

- 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)

## 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

## 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 accepts X
- Conclusion: Solving SAT in poly time implies solving any problem in NP in poly time

## 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!

# P=NP?

- Biggest open question in CS
- Can NP-complete problems be solved in polynomial time?
- Probably not, but nobody has been able to prove it yet
- 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

# Generalization

- Show problem A is NP-hard because known NP-hard problem B is a special case of A
- Example: SAT generalizes 3SAT
  - Every valid 3SAT instance is a valid SAT instance
  - A poly-time SAT solver would, therefore, ALSO be a poly time 3SAT solver
  - Conclusion: SAT is at least as hard as 3SAT: NP-hard
- How does this relate to reductions?

# 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

## k-clique -> Subgraph Isomorphism

- k-clique: Given G=(V,E), dthere 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)
- On board

## 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 optimization to decision**

## Weak vs. Strong Hardness

- Some problems can be brute-forced if the range of numbers involved is not large (note: range is exponential in input size)
- Subset sum: $\exists$ subset of a group of natural numbers that sums to k?
  - Use dynamic programming
  - Answer question for 1…j
  - Build answer for j+1 from answers to 1…j
  - Build up to k
- Such problems are weakly NP-hard

## What's harder still?

- P-space hardness
- Algorithms in P-space require polynomial space
- Why is this at least as hard as P-time?
- Still harder: exp-time

## 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