P versus NP, NP hard and NP complete

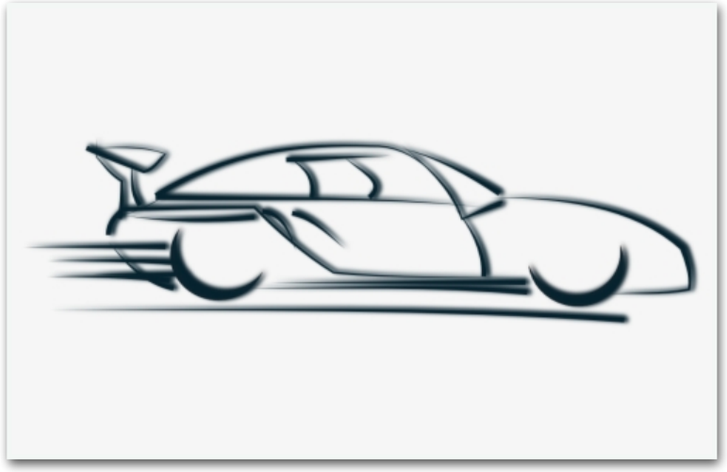
Admin

- Assignment 7 notes:
    - 3 Problems on Flow Networks
        - Increasing order of difficulty
        - Two ask for “an efficient algorithm to solve…”
            - Describe how you would construct a flow network (define **all** vertices, edges, capacities)
            - Describe how you would use your flow network to answer the actual question (e.g., Given a network G constructed as described above, we would use FF to calculate the max flow of G. If the max flow is less than g, then the Angels are eliminated from contention.)
            - Give/analyze the run time (otherwise, how would you justify your claim that it is an efficient algorithm?)

Today: Overview and Classes P and NP

Some Algorithms/Problems So Far

- Single source shortest paths—can find shortest path from vertex to all other vertices using Dijkstra's algorithm. We improved the runtime to $O(m \log n)$ with the union-find data structure.
- Weighted interval scheduling—can try all combinations in $O(2^n)$, but can solve in $O(n^2)$ using Dynamic programming
- Network flow seems very difficult to solve, but we saw how to solve it in $O(nmC)$ (and noted it possible to solve in $O(n^2m)$)



We've explored techniques to analyze algorithms and make algorithms faster!

Shifting Focus

Most of the class has been tools to efficiently solve problems (and prove the efficiency/correctness of solutions)

- Now we're going to shift to higher-level questions:
  - What problems can a computer solve efficiently?
  - What problems can't* a computer solve efficiently?

Efficiency: Polynomial time

When we say efficient, what we really mean is polynomial in the size of the input.

So our questions from before become:

- What problems can a computer solve in polynomial time?
- What problems **can't*** a computer solve in polynomial time?

(*probably cant)

Technical Setup

We will focus on **decision problems** — problems with a yes or no answer. Examples include:

- Does this directed graph have a topological order?
- Is this graph bipartite?
- Do these two strings have an Edit Distance less than 10?
- Does this graph have a perfect matching?

Technical Setup

With care, we can craft a decision analog to most problems

- Instead of "*Find the flow of this network*", we can ask:
  - *Does this network have a valid flow of at least* k*?*"
    - "*Find the optimal schedule of these intervals*" becomes:
      - "*Can we schedule at least* k *intervals*?"
- These are (essentially) the same! Why?
  - We can always binary search for the optimal value!

Technical Setup

Decision problem means that every solution is “yes” or “no”

- Yes instances can represented as a set of inputs A
  - $x \in A$ means that the solution to x is “yes”
  - $x \notin A$ means that the solution to x is “no”

- So can have (for example): A is the set of all flow networks which permit flow at least k
- Or can have: A is the set of all pairs of strings (a, b) where the edit distance between a and b is at most k
- If we define s to be the encoding of the problem input as a string, we can ask: does s belong to the set A? (We’ll revisit this later)

Class P

Class P

P: the class of problems that can be **solved** in polynomial time

- Edit distance is in **P**
- Max flow is in **P**
- Bipartite matching is in **P**
- Knapsack?
  - The dynamic programming algorithm we saw is pseudo-polynomial! So we don't know yet…

Class NP

Class NP—Intuition

NP is the class of problems that can be ***verified*** in polynomial time

- What do we mean by **verify**?
    - If I give you helpful information, say a proposed solution, you can easily check that it is correct
    - Note, *verifying* a solution is very different than *finding* a solution!

Class NP—Intuition

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		

☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

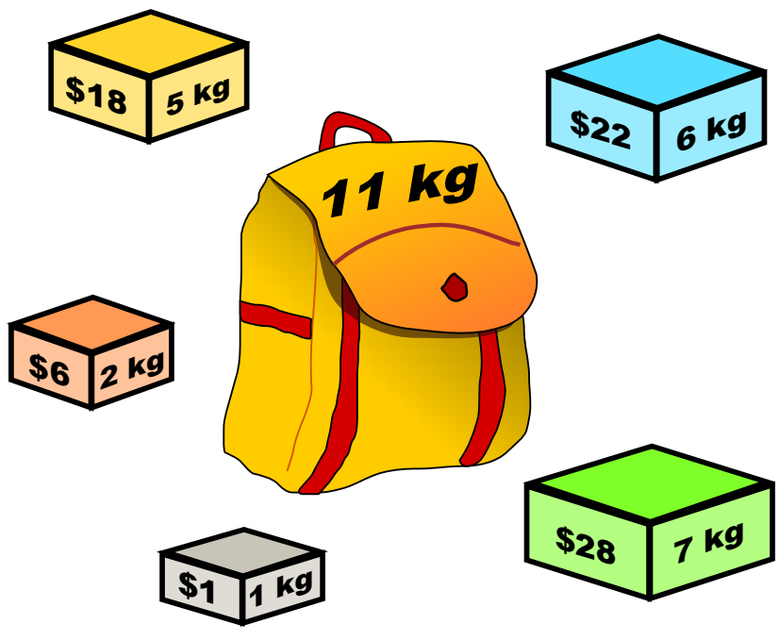
Sudoku is easy if I give you information (e.g., giving you the solution). So sudoku is in **NP**

Class NP—Intuition

- Example (Knapsack capacity C = 11)
  - {3, 4} has value $40 (and weight 11)

i	v_i	w_i
1	$1	1 kg
2	$6	2 kg
3	$18	5 kg
4	$22	6 kg
5	$28	7 kg

knapsack instance
(weight limit W = 11)



Knapsack is easy if I give you information (e.g., giving you the solution). So knapsack is in NP

Class NP: Formally

Let s be the input to a problem encoded as a binary string, and let $|s|$ be the length of that string.

We will define a decision problem X with the set of strings for which the answer is “yes”.

Verifiers. Algorithm $V(s, c)$ is a verifier for problem if, for every input string $s \in X$, there exists a certificate c, such that $V(s, c) =$ yes iff $s \in X$.

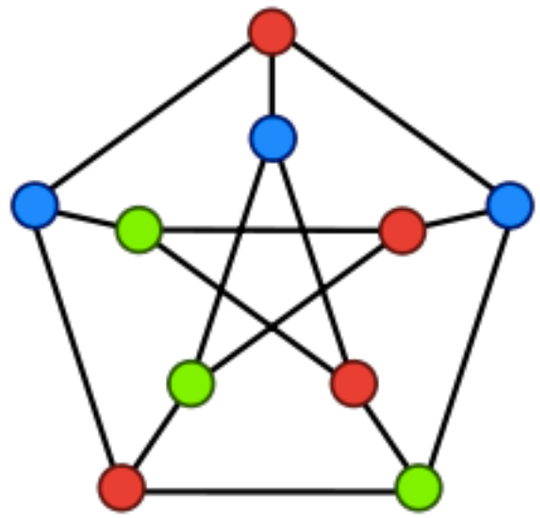
Definition. NP = the set of decision problems for which there exists a polynomial-time verifier. That is,

- $V(s, c)$ is a polynomial time algorithm
- Certificate c is of polynomial size:
    - $|c| \leq p(|s|)$ for some polynomial $p(.)$

Graph-Coloring $\in$ NP

Graph-Coloring. Given a graph $G = (V, E)$, is it possible to color the vertices of G using only three colors, such that no edge has both end points colored with the same color?

- Graph-Coloring $\in$ NP
  - **Input**: a set of vertices, colors, and edges
  - **Certificate:** assignment of colors to vertices
  - **Poly-time verifier:** check for each edge if ends points same color or not, and check that at most 3 colors used

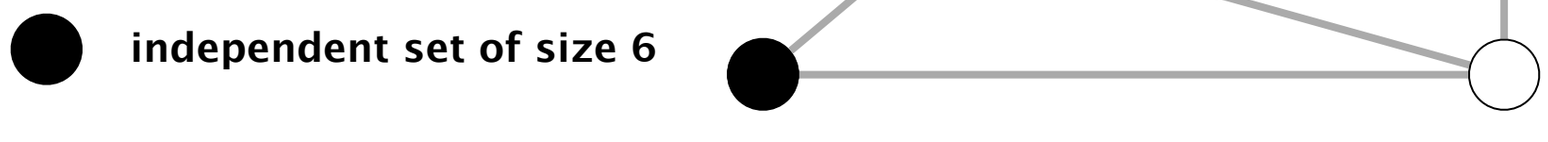


A 3-colorable graph

Independent Set

- Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent, that is, for any $x, y \in S$, $(x, y) \notin E$

- **IND-SET Problem.**
  Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?



IND-SET $\in$ NP

- Given a graph $G = (V, E)$, an independent set is a subset of vertices $S \subseteq V$ such that no two of them are adjacent, that is, for any $x, y \in S$, $(x, y) \notin E$

- **IND-SET Problem.** Given a graph $G = (V, E)$ and an integer k, does G have an independent set of size at least k?

- **IND-SET** $\in$ NP.
  - **Certificate:** a subset of vertices
  - **Poly-time verifier:** check if any two vertices are adjacent and check if size is at least k

Quick Question

- Is **P** ⊆ **NP**?
    - If a problem is in **P**, does that mean that it is in **NP**?
- Yes! If a problem can be solved in polynomial time, it can be verified in polynomial time.
- How?
    - Just solve directly (Can just set $c =$ ””)

Satisfiability

- The next problem is the classic example used when listing examples of problems in **NP**
  - (and, as we'll soon see, probably not in **P**)
- There are also many different small variations on the same problem (we'll see a couple)
- **Idea**: given a logical equation, can we assign "true" and "false" to the variables to satisfy the equation?

SAT, 3SAT $\in$ NP

- **SAT**. Given a conjunctive normal form (CNF) formula ϕ, does it have a satisfying truth assignment?
- **3SAT.** A SAT formula where each clause contains exactly 3 literals (corresponding to different variables)

- $\phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$
- Satisfying instance: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$, where 1 : true, 0 : false
- SAT, 3-SAT $\in$ NP
    - **Certificate**: truth assignment to variables
    - **Poly-time verifier**: check if assignment evaluates to true

P versus NP

P vs NP

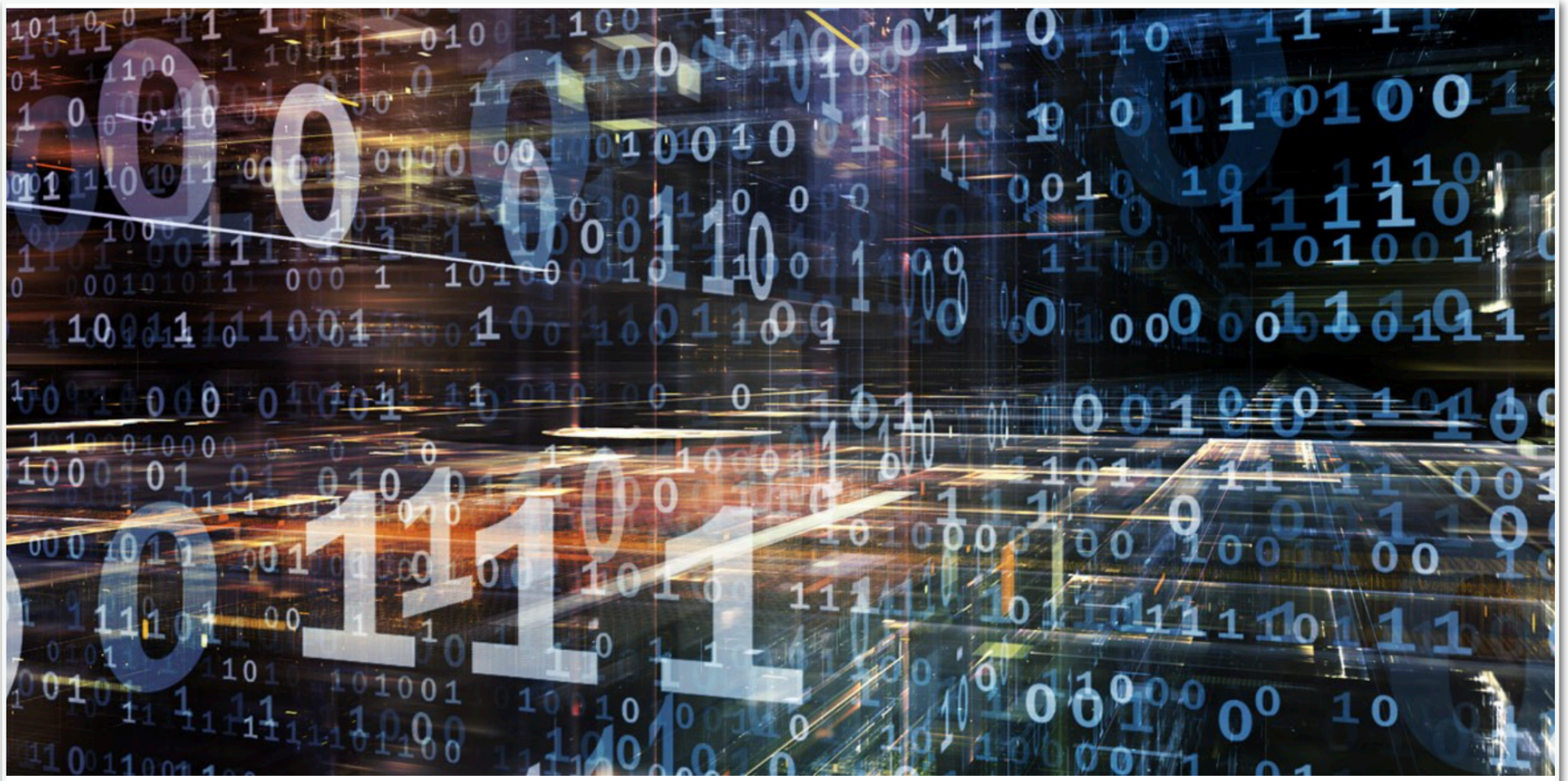
- We know that every problem in **P** is also in **NP**
  - If we can solve it in polynomial time, we can verify it
- What about the reverse? That is to say:
  - If a problem can be efficiently *verified*, does that mean it can be efficiently solved in the first place?
  - Or, do there exist problems that can be verified quickly that are *impossible* to solve quickly?

Why Do We Care?

- If $\mathbf{P = NP}$, there are many consequences:
  - Lots of important problems can be solved quickly!
  - Can build things better, faster, more efficiently
    - (Public key) cryptography does not exist 😱
- If $\mathbf{P \neq NP}$:
  - Many problems have no efficient solutions
    - And we can stop trying to solve them faster!

Many very smart people believe that $\mathbf{P \neq NP}$

Million Dollar Question:
P vs NP



P vs NP and the $1M Millennium Prize Problems

What's the most difficult way to earn $1M US Dollars?

https://medium.com/@mpreziuso/

Million Dollar Question:
P vs NP

- The biggest open problem in computer science
- One of the biggest in math as well
- We are not even close to solving it!
  - Every so often, a new proof will be proposed
  - So far, all of these proofs have had flaws…

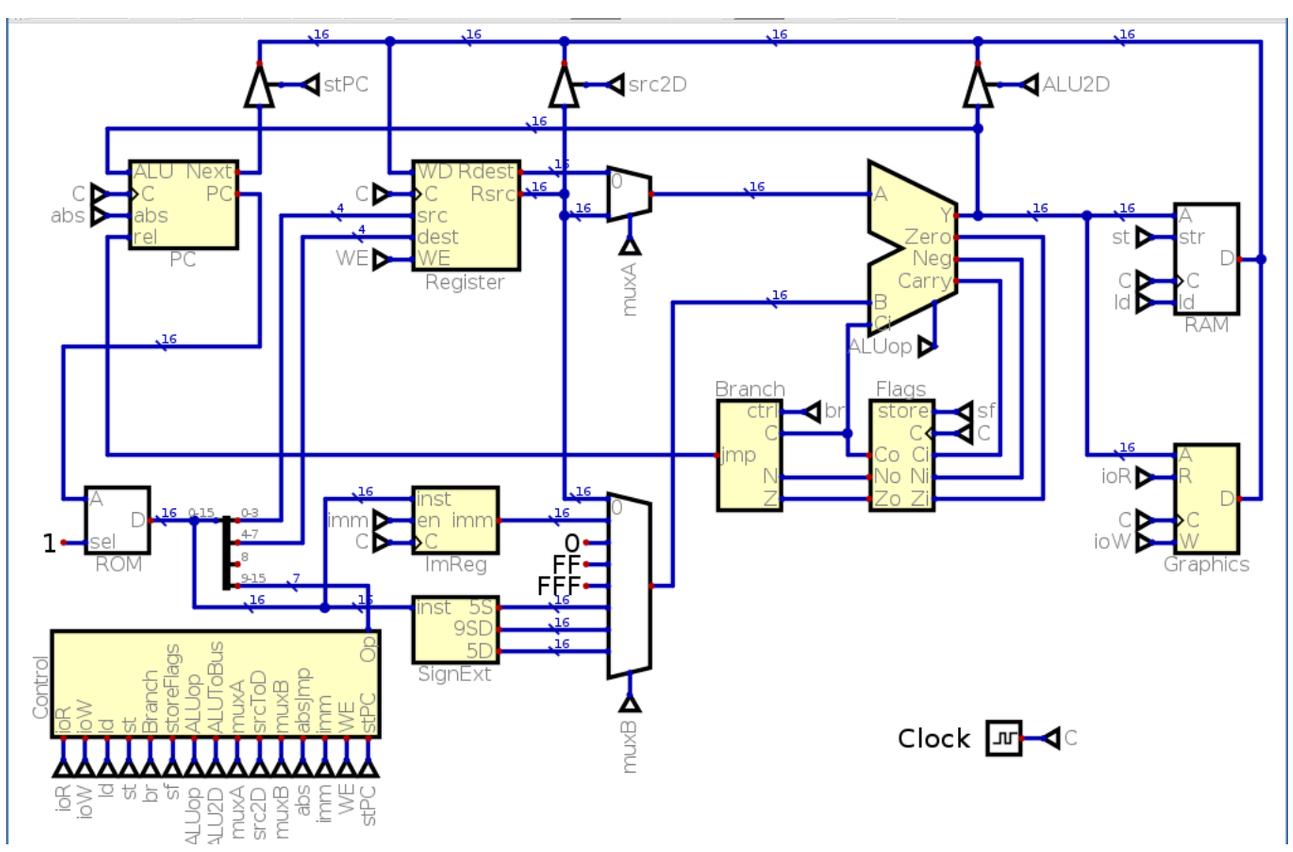
NP-hard and NP-Complete Problems

Cook-Levin Theorem

- If **SAT** can be solved in polynomial time, then *any* problem in **NP** can be solved in polynomial time
- So if **SAT** can be solved in polynomial time, then **P = NP**
- How is this possible?

Cook-Levin Theorem

- Idea: any computer program can be represented by a circuit.
- If we can solve **SAT** in poly time, we can figure out the answer given by the circuit for an NP problem in poly time



You'll see the proof in CS 361

We will wave our hands in CS 256…

NP-Hard Problems

- A problem X is **NP-hard** if:
  - If X can be solved in polynomial time, then any problem in **NP** can be solved in polynomial time
  - That is, if X can be solved in polynomial time, then **P = NP**

NP-hard problems are at least as hard as every problem in NP

What Does This Mean?

- We think that, probably, **P** ≠ **NP**
- So if a problem is **NP**-hard, then you probably cannot obtain a polynomial-time algorithm for it

Classifying Problems as Hard

- We are frustratingly unable to prove a lot of problems are **impossible** to solve efficiently
- Instead, we say problem X is likely very hard to solve by saying, *if a polynomial-time algorithm was found for* X*, then something we all believe is impossible will happen*
- This is what we mean when we say X is NP-hard: if $X \in \mathsf{P}$, then $\mathsf{P} = \mathsf{NP}$

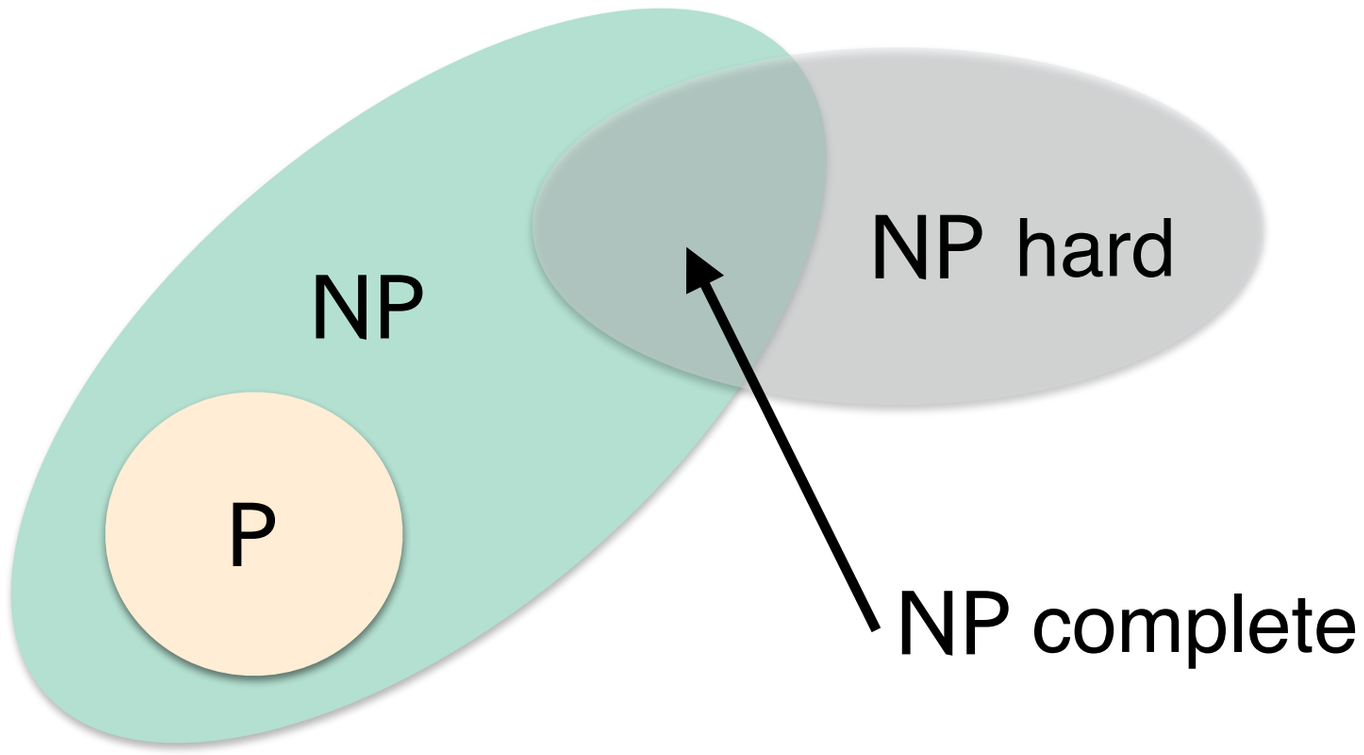
Classifying Problems as Hard

- Instead, we say problem X is likely very hard to solve by saying, *if a polynomial-time algorithm was found for X, then something we all believe is impossible will happen*
- This is what we mean when we say X is NP-hard: if $X \in$ P, then P = NP

- (Erickson) Calling a problem **NP** hard is like saying, *"If I own a dog, then it can speak fluent English"*
  - You (probably) don't know whether or not I own a dog, but you are supremely confident I don't own *a talking dog*.
  - Why though? You don't have concrete evidence, other than years of looking at other dogs that do not speak.
- So, if a problem is **NP** hard, no one should believe it can be solved in polynomial time, even though we don't have proof…

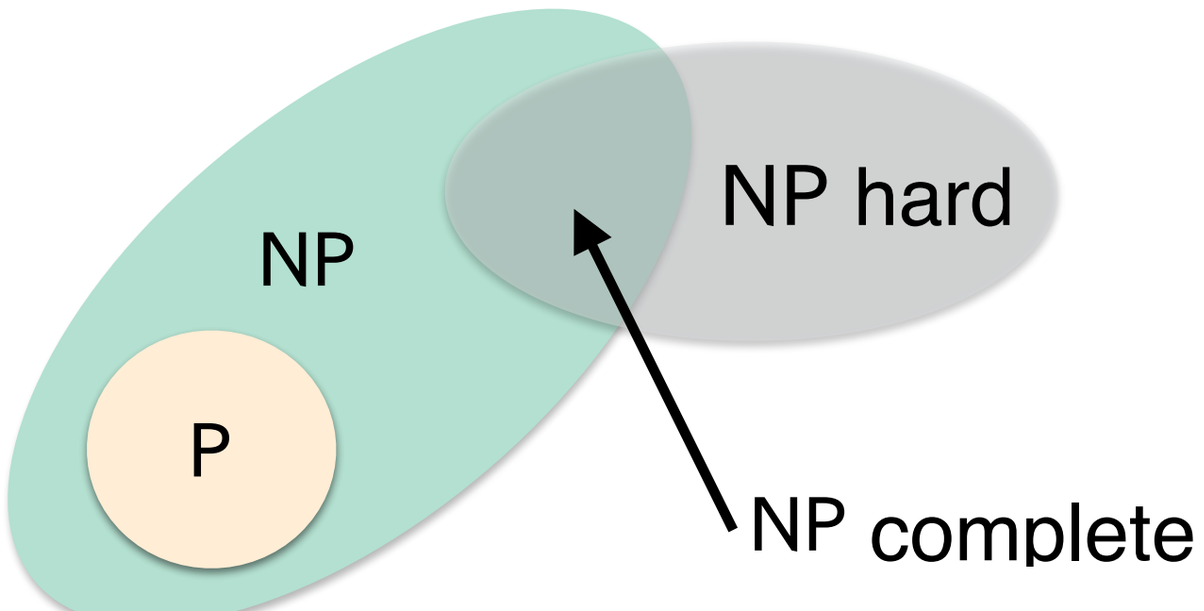
NP Completeness

- **Definition.** A problem X is NP complete if X is NP hard and $X \in$ NP
- SAT is **NP** complete
    - SAT $\in$ **NP**: given an assignment to input gates (certificate), can verify whether output is one or zero in poly-time
    - SAT is **NP** hard (Cook-Levin Theorem)



Polynomial-Time Reducibility

- (**KT Definition**): A problem X is NP complete if X is NP hard, and for all $Y \in$ NP, $Y \leq_p X$
    - $Y \leq_p X$ means that Y can be reduced to X in polynomial time
    - Said differently, we can map (in polynomial time) a general instance of a problem Y to a specific instance of a problem X, where a solution to X yields a solution to Y
- Thus, NP-complete problems are the hardest problems in NP



Acknowledgments

- Some of the material in these slides are taken from
  - Shikha Singh
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)