# CS340 - Analysis of Algorithms

NP and Computational Intractability

## Announcements:

**HW9 Released - Due December 8th (Next Monday)**

Final Exam Options:

1. Dec 12 1-4pm Park 230
2. Dec 15 9:30-12:30 Park 159

# Efficient Algorithms - Polynomial Time

- Up until now we've studied algorithms for a wide range of problems and informally categorized them (greedy, divide and conquer, DP, network flow)

- We aimed for algorithms in polynomial time

**Table 2.1** The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds $10^{25}$ years, we simply record the algorithm as taking a very long time.

| | $n$ | $n \log_2 n$ | $n^2$ | $n^3$ | $1.5^n$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|---|
| $n = 10$ | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 4 sec |
| $n = 30$ | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 18 min | $10^{25}$ years |
| $n = 50$ | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 11 min | 36 years | very long |
| $n = 100$ | < 1 sec | < 1 sec | < 1 sec | 1 sec | 12,892 years | $10^{17}$ years | very long |
| $n = 1{,}000$ | < 1 sec | < 1 sec | 1 sec | 18 min | very long | very long | very long |
| $n = 10{,}000$ | < 1 sec | < 1 sec | 2 min | 12 days | very long | very long | very long |
| $n = 100{,}000$ | < 1 sec | 2 sec | 3 hours | 32 years | very long | very long | very long |
| $n = 1{,}000{,}000$ | 1 sec | 20 sec | 12 days | 31,710 years | very long | very long | very long |

# Efficient Algorithms - Polynomial Time

- Can some problems not be solved in polynomial time?
  - Are some problems just too hard?
  - For a certain set of problems, we do not know of any polynomial-time algorithms, **but we cannot prove that no polynomial-time algorithm exists**
- It *has* been proved that some of these problems are equivalent
  - A polynomial-time algorithm for one of these problems would imply the existence of a polynomial-time algorithm for all of them
- These are called **NP-Complete** problems
  - All of these open ended questions are really a single open question and a single class of complexity we don’t fully understand

# Complexity Classes

- **P** is the set of decision problems for which there are known polynomial time solutions

- **NP** is the set of decision problems whose solution can be *verified* in polynomial time
  - “NP” stands for nondeterministic polynomial time

- **P ⊆ NP**

- **Does P = NP? Does P ≠ NP?**

# Complexity Classes

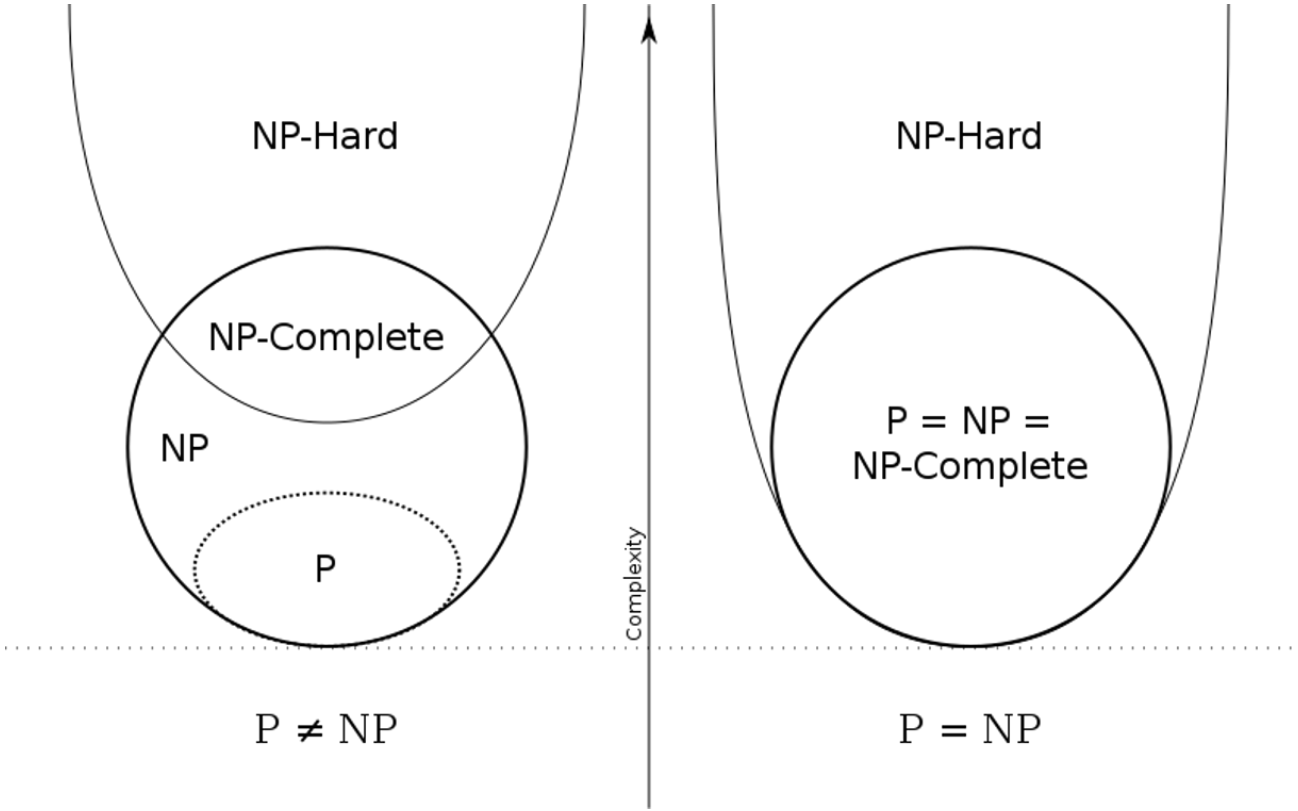
- **NP-Complete (NPC):** A problem *p* in NP is NP-Complete ($p \in$ NPC) if a polynomial time solution to *p* would lead to a polynomial time solution for all NP
  - Any NP problem can be reduced to a $p \in$ NPC in polynomial time

- **NPC $\subseteq$ NP**

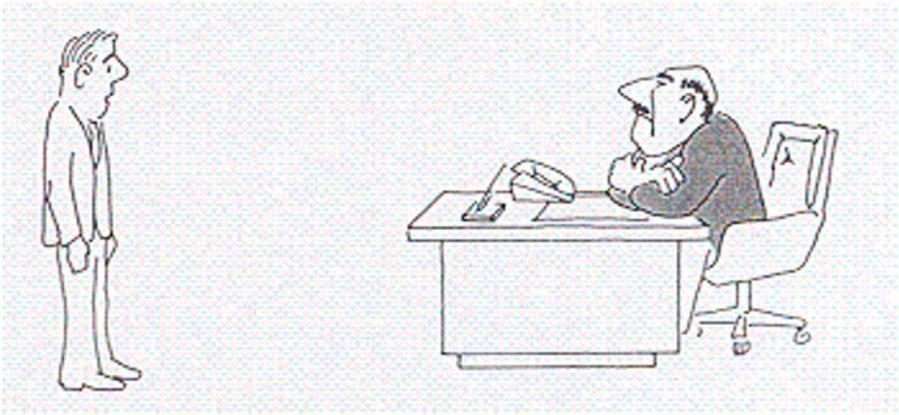
# Complexity Classes

- **NP-Hard**: A problem *h* is in NP-Hard if, for every problem $j \in$ NP, *j* can be reduced to *h* in polynomial time
  - At least as hard as the hardest problems in NP
  - Does not require *h* being in NP
  - Does not require *h* being a decision problem

# Classes of Complexity



# P = NP 😖



I can't find an efficient algorithm, I guess I'm just too dumb.

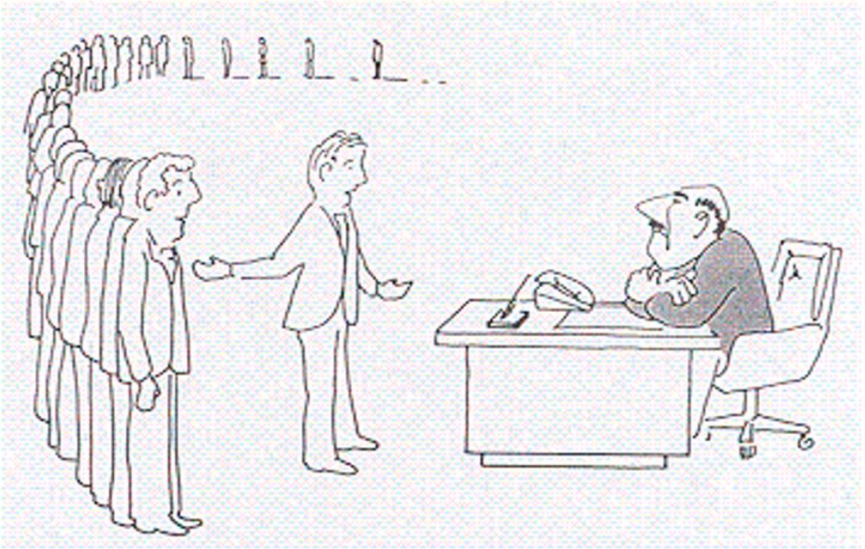
from *Computers and Intractibility*
by Garey and Johnson, 1979

# P ≠ NP !

I can't find an efficient algorithm because no such algorithm is possible!

from *Computers and Intractibility*
by Garey and Johnson, 1979

# P != NP ??



I can't find an efficient algorithm, but neither can all these famous people.

from *Computers and Intractibility*
by Garey and Johnson, 1979

# Polynomial Time Reductions

- We will use reductions to compare relative difficulty of different problems
- “Problem B is at least as hard as problem A”
- If we had a “black box” capable of solving B, then we could also solve A

Can arbitrary instances of problem A be solved in a polynomial number of calls to a black box that solves problem B?

If yes, we say $A \leq_p B$ or A is polynomial-time reducible to B

- B is at least as hard as A
- A is no harder than B

# Polynomial Time Reductions

$A \leq_p B$ : arbitrary instances of problem A be solved in a polynomial number of calls to a black box that solves problem B.

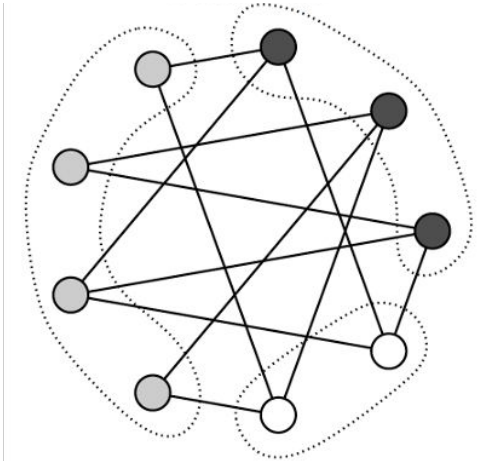
- B is at least as hard as A
- A is no harder than B

- If $A \leq_p B$ and $B \in P$ , then $A \in P$

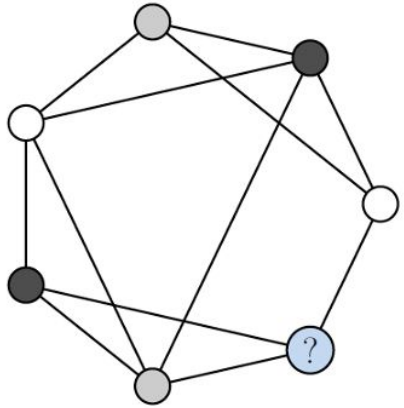
- If $A \leq_p B$ and $A \notin P$ , then $B \notin P$

# 3-Colorability

Given a graph G, can each of its vertices be labeled with one of three different colors, such that no two adjacent vertices have the same color?



3-colorable? ✅



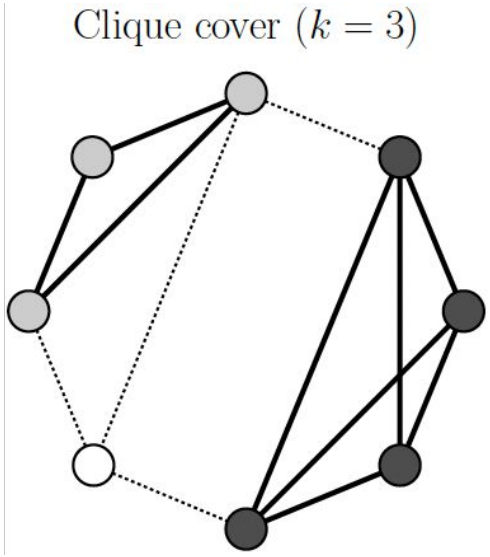

3-colorable? ✖

3Col ∉ P

# Clique Cover

- Given a graph G = (V,E), we say that a subset of vertices V’ $\subseteq$ V forms a *clique* if for every pair of distinct vertices u,v $\in$ V’, (u,v) $\in$ E

- Given a graph G = (V,E), and an integer k, can we partition the vertex set into k subsets of vertices $V_1$, ..., $V_k$ such that each $V_i$ is a clique of G?

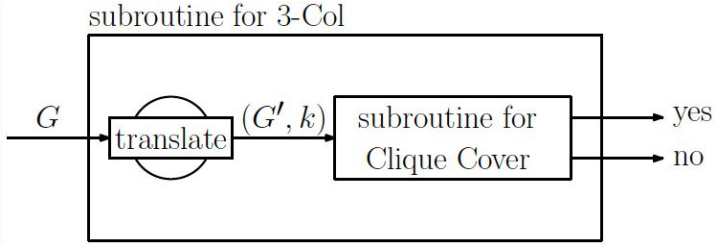


# Reduction from 3-Colorability to Clique Cover

$A \leq_p B$ : arbitrary instances of problem A be solved in a polynomial number of calls to a black box that solves problem B.

- A is reducible to B
- B is at least as hard as A (A is no harder than B)
- If $A \notin P$ , then $B \notin P$

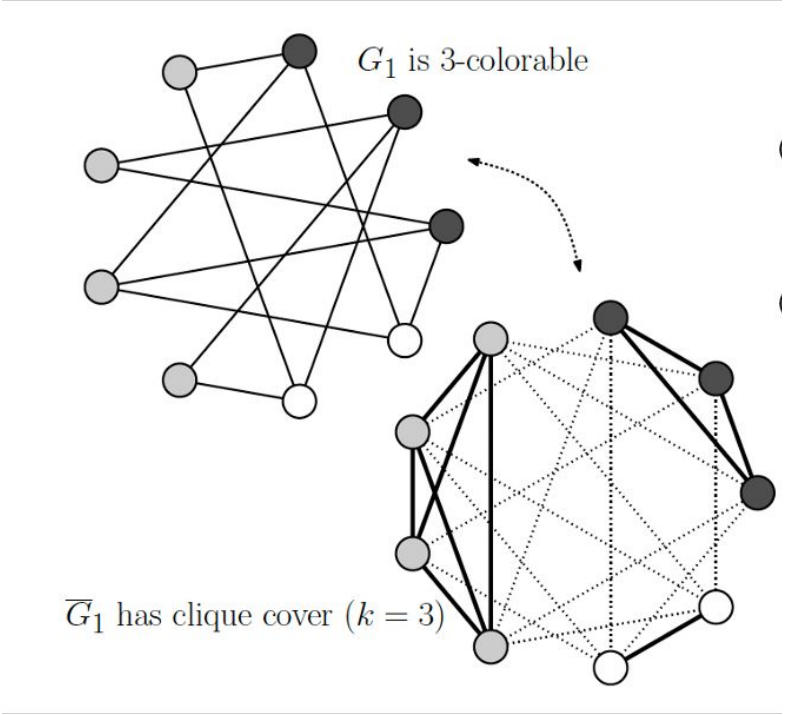
- A = 3-Colorability (3Col). $A \notin P$
- B = Clique Cover (CCov)
- **To show that CCov $\notin$ P, Find a reduction that maps an instance G for 3Col into an instance G' for CCov**
- 3Col $\notin$ P and 3Col can be solved with (a polynomial number of) calls to a solver for CCov
- So, 3Col is not harder than CCov
- Therefore, Clique Cover $\notin$ P

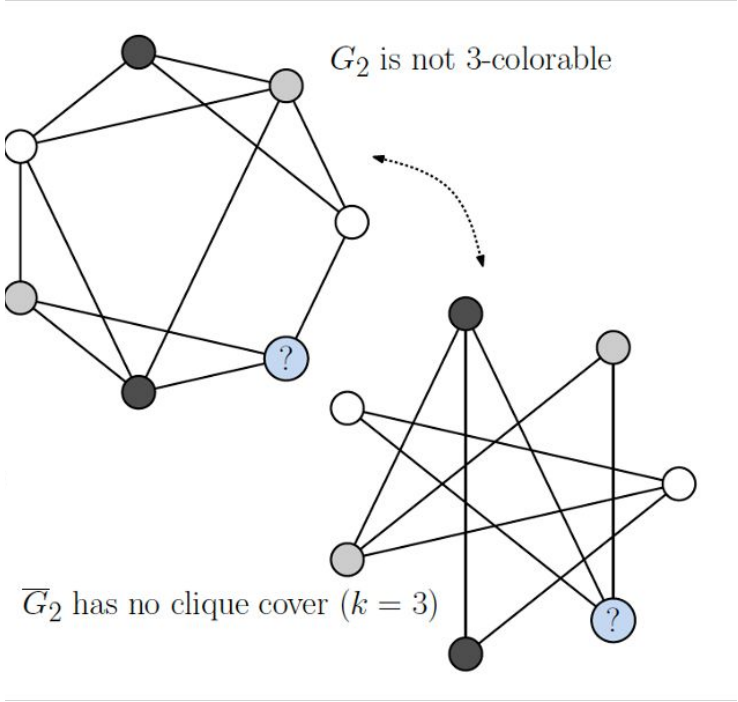


# Reduction from 3-Colorability to Clique Cover

- Find a reduction that maps an instance G for 3Col into an instance G' for CCov
- Let G be a graph input to 3Col
- Given a graph G = (V, E), construct the complement graph $\overline{G}$, where two distinct nodes are connected by an edge iff they are not connected in G
- $(\overline{G}, 3)$ becomes the input to CCov, and we use the "black box"
- `CCov(`$\overline{G}$`, 3)`

# Reduction from 3-Colorability to Clique Cover



# Reduction from 3-Colorability to Clique Cover

**Proof:** G = (V,E) is 3-colorable iff its complement $\overline{G}$ = (V, $\overline{E}$) has a clique cover of size 3

□ If G is 3-colorable, then let $V_i$, 1 ≤ i ≤ 3, be the three vertex classes. This is a clique cover of size 3 for $\overline{G}$ since u,v ∈ $V_i$ □ (u,v) ∉ E □ (u,v) ∈ $\overline{E}$

□ If $\overline{G}$ has a clique cover of size 3, denote them $V_i$ . For i = 1,2,3 give the vertices of $V_i$ color i. This is a valid coloring in G because u,v ∈ $V_i$ □ (u,v) ∈ $\overline{E}$ □ (u,v) ∉ E

# Polynomial Time Reduction

If $A \leq_p B$, “A is polynomial-time reducible to B”

A can be solved with polynomial time calls to a black box that solves B

B is at least as hard as A
A is no harder than B

- Reduce to the problem you want to show is harder / not easier
- If $A \notin P$ and we want to show $B \notin P$, we reduce A to B ($A \leq_p B$).
- If you want to show $B \in NP$, give a polynomial time verifier for it
- If you want to show $B \in$ NP-Hard, reduce an $A \in$ NPC problem to B ($A \leq_p B$)
- If you want to show $B \in$ NPC, reduce an $A \in$ NPC problem to B ($A \leq_p B$) and show $B \in$ NP

# SAT - An NPC Problem

- A boolean formula consists of variables (say $x$, $y$ and $z$) and logical operators *not* ($\bar{x}$), *and* ($x \wedge y$), *or* ($x \vee y$).
- Given a boolean formula, we say that it is *satisfiable* if there is a way to assign truth values ($0$ or $1$) to the variables such that it evaluates to $1$.
- $\big(x \wedge (y \vee \bar{z})\big) \wedge ((\bar{y} \wedge \bar{z}) \vee \bar{x})$
- $(\bar{z} \vee x) \wedge (z \vee y) \wedge (\bar{x} \wedge \bar{y})$

SAT or UNSAT?

- SAT {X →1, Y →0, Z →1 }
- UNSAT

# Conjunctive Normal Form (CNF)

Each *clause* is separated by an AND operator

$C_1 \wedge \ldots \wedge C_n$

And each *clause* and is of the form:

$L_i \vee \ldots \vee L_m$

where each $L_i$ is called a *literal* and is either a boolean variable or a negation of the variable

# Conjunctive Normal Form (CNF)

**Example** 6.A The following is a CNF formula with two clauses, each of which contains two literals:

$$(p \vee \neg r) \wedge (\neg p \vee q)$$

The following formula is *not* in CNF:

$$(p \wedge q) \vee (\neg r)$$

■

# 3SAT

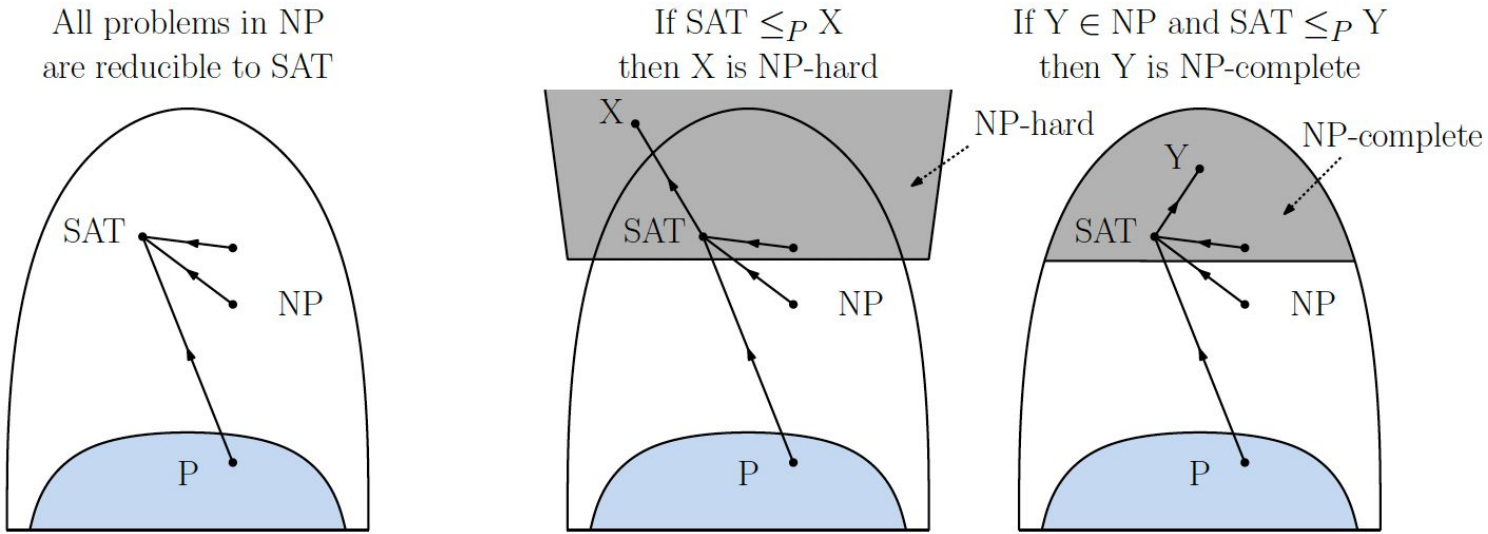
- CNF SAT formula with exactly 3 clauses and each clause is the boolean-or of exactly 3 literals

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3 \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$

- 3SAT is NPC

# Cook's Theorem

SAT is the first problem that was proven to be NPC



# Summary

- Today we introduced the notion of Intractability
  - Problems that we don’t know polynomial time algorithms for
- Complexity classes: P, NP, NPC, and NP-Hard
- Reductions can be used to compare difficulty of problems
- We did a reduction from 3-Coloring to Clique Cover to show Clique Cover $\notin$ P
- We introduced an NPC problem SAT
- **HW9 due one week from today**