CMSC 451 - Algorithm Design
Lecture 15 - NP-Completeness: Basic Definitions

So far...

- How to design and analyze efficient algorithms for combinatorial problems
- What do we mean by efficient?

- Polynomial time - $O(n^c)$, where c = constant (worst case)
- Exponential time - $\Omega(c^n)$, $c > 1$

Hard Problems:

Near end of 1960's:

- Poly-time algorithms for many problems
- Many problems defied poly-time solutions
  - Best solutions based on brute-force
    $\Rightarrow$ Exponential time

- Researchers sought the crucial property that distinguishes between them

- Spoiler alert - They failed!

Enter Stephen Cook (1971)

- For many combinatorial problems, the best solution has the following form:
  - guess the answer + verify it is correct

- NP-Complete - The "hardest problems" in NP

> If you can solve any of these efficiently - you can solve every problem in NP efficiently.

NO WAY!

- Boolean satisfiability (SAT) is NP-complete

Enter Richard Karp (1972)

- Showed that 21 well-known problems are NP-Complete - This is major!

  - Travelling salesman
  - Hamiltonian cycle
  - Clique
  - Independent set
  - Graph coloring
  - ⋮

- Is P = NP? $1M prize for the answer

Decision Problems + Language Recognition

- It will simplify the theory to assume all problems have same output - yes or no → Decision problem

- Optimization: Find min. spanning tree for G
- Decision: Given $G + z \geq 0$, does G have a spanning tree of weight $\leq z$?

Language Recognition

- For historical reasons (automata theory predates algorithm theory) NP problems are expressed as language recognition

- Assume we have function serialize(I) - Encodes input I (e.g., graph, set, number) as a string

Decision problems	→	Language recognition
Does G have an MST of weight $\leq z$?	→	MST = $\{(G, z) \mid$ G's MST has weight $\leq z\}$

Complexity Class P:

Decision Problems

Def: P is the set of languages such that membership can be determined in polynomial time.

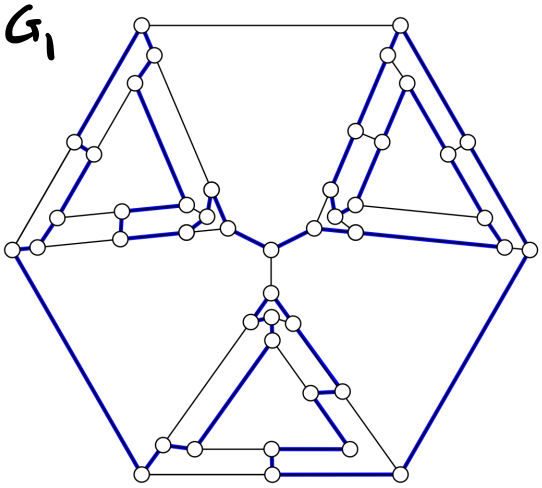
Eg. Can solve MSTs efficiently, so MST $\in$ P

Are there problems (languages) not in P?

Hamiltonian Cycle: Given a graph G, a Hamiltonian cycle is a simple cycle that visits each vertex of G (exactly once)

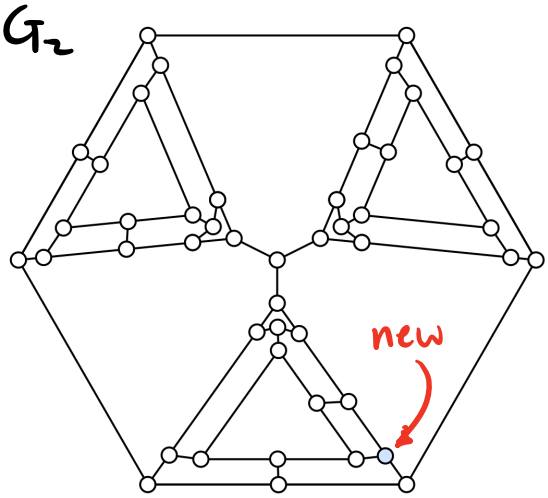
$HC = \{ G \mid G \text{ has a Ham. cycle} \}$

G_1



$G_1 \in HC$

G_2



$G_2 \notin HC$? (don't know)

Is $HC \in P$? (Not known to science)

HC has a nice property - verifiability

- If $G \in HC$, it is easy to verify this
  (Show me the cycle, and I'll check)
- However, if $G \notin HC$, how to verify ???
  (No idea)

Complexity Class NP:

> **Def:** NP is the set of languages such that membership can be verified in poly time.

What do you mean by "verified"?

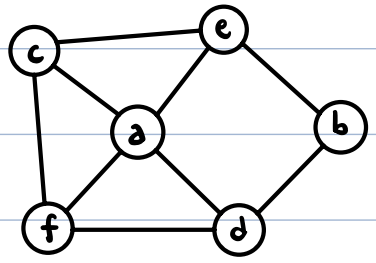
Given a language L (set of strings)

- Given string x, want to know whether $x \in L$.
- A certificate is a "helper string" y:
  - if $x \in L$, y can be used to prove this in polynomial time
  - if $x \notin L$, then we don't care

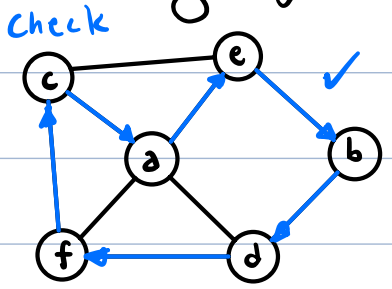
HC = { G | G has a Hamil. Cycle }

Certificate = List of vertices forming cycle

x:



y: ⟨c, a, e, b, d, f⟩



This is weirdly assymetric!

- if $x \in L$ - need to verify
- if $x \notin L$ - don't care

That is just how it is defined.

Are their (natural) languages that are not poly-time verifiable?

UHC = { G | G has a unique Ham. cycle }

If $x \in$ UHC, you can show one Ham. cycle, but how do you prove there no other?

How to show P ≠ NP? Strategy:

- Find the hardest problem in NP
- Show that this problem is not in P
- Such a problem will be called NP-complete

Before defining NP-Completeness, we need to discuss reductions.

Polynomial-Time Reductions:

Motivation: How do we show that problem is hard to solve?

Consider:

H - a well-known (very likely) hard problem - Many experts tried + failed

U - your problem, which you suspect may be hard (but who trusts you?)

Want to show:

$H \notin P \Rightarrow U \notin P$

$H \notin P$: what everyone believes

$U \notin P$: what you want to prove

Let's prove contrapositive!

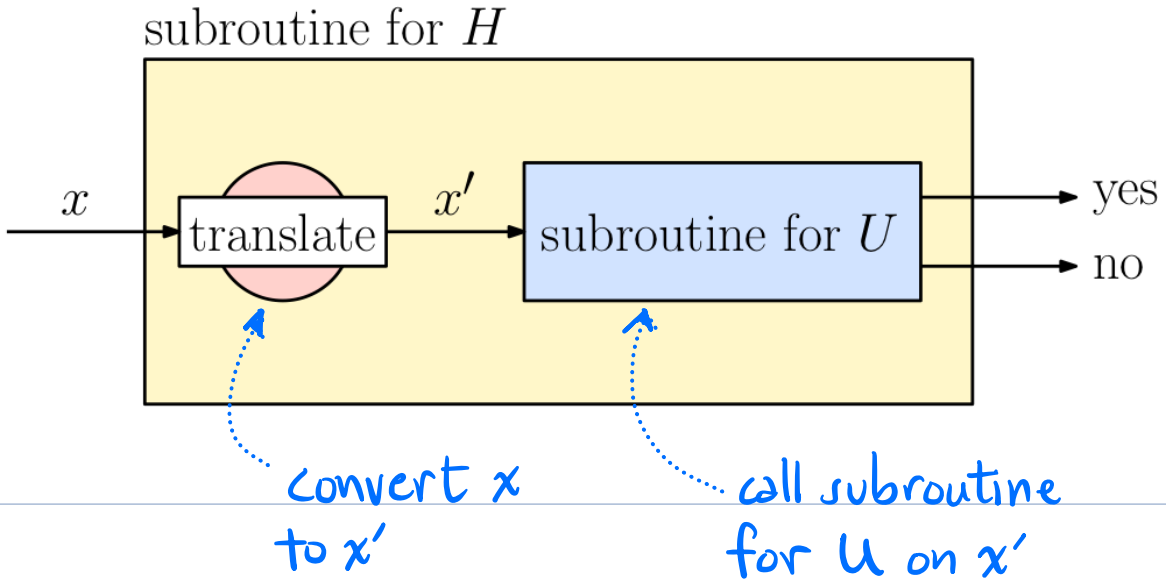
$U \in P \Rightarrow H \in P$

$U \in P$: If I could solve my problem

$H \in P$: Then I could solve a famous hard problem ← NO WAY!

How? Use a subroutine for my problem U (as a black box) to solve H

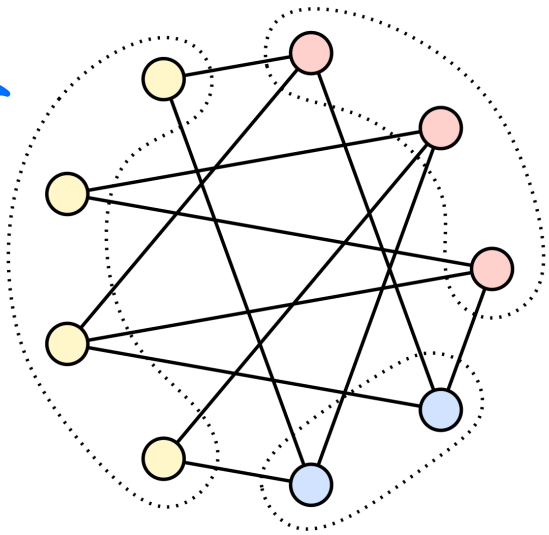
Pictorially: Is $x \in H$?



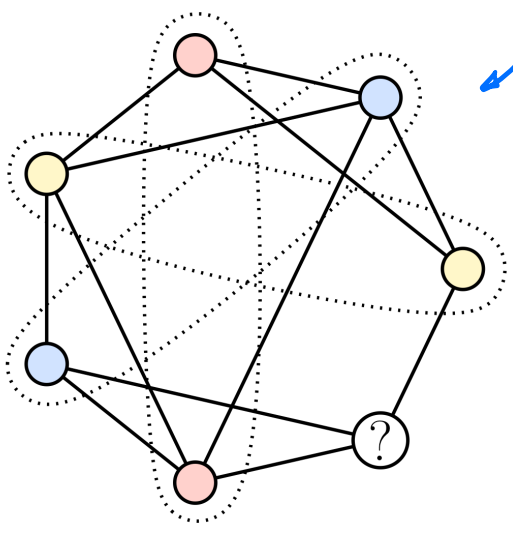
Example: Let's try this on two actual problems.

Hard: 3-Coloring (3COL) - Given a graph G can its vertices be labeled with one of three colors, so that no two adjacent vertices have same color.

3-Colorable



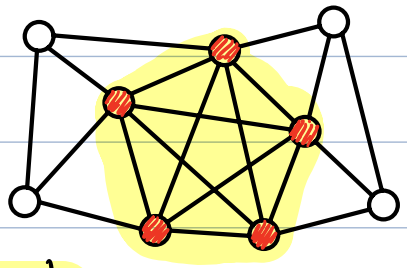
Not 3 colorable (Need at least 4)



No know poly-time algorithm - even for planar graphs

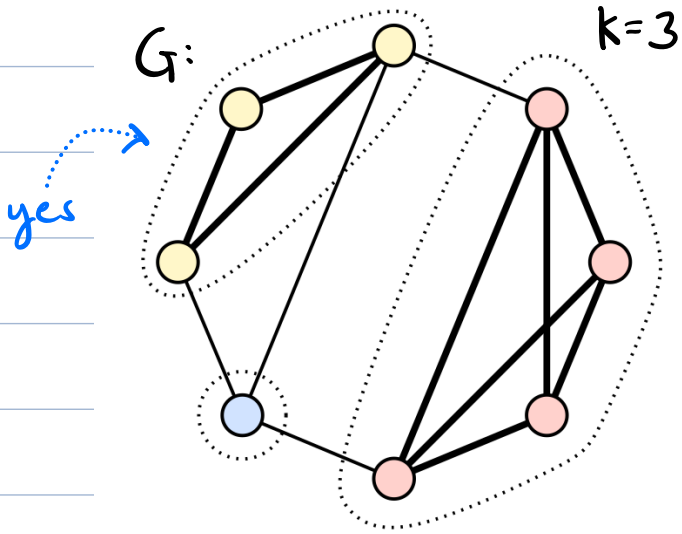
Your Problem (which you want to show is hard)

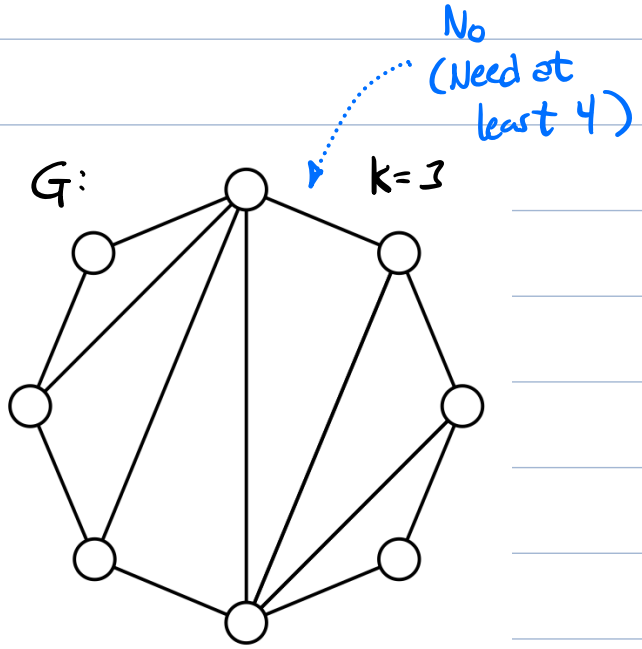
Def: A clique is a completely connected subgraph



Clique Cover (CCov): Given a graph $G=(V,E)$ and integer k, can we partition vertices into k sets $V_1, V_2, \ldots, V_k$ such that each V_i is a clique in G.

Example:





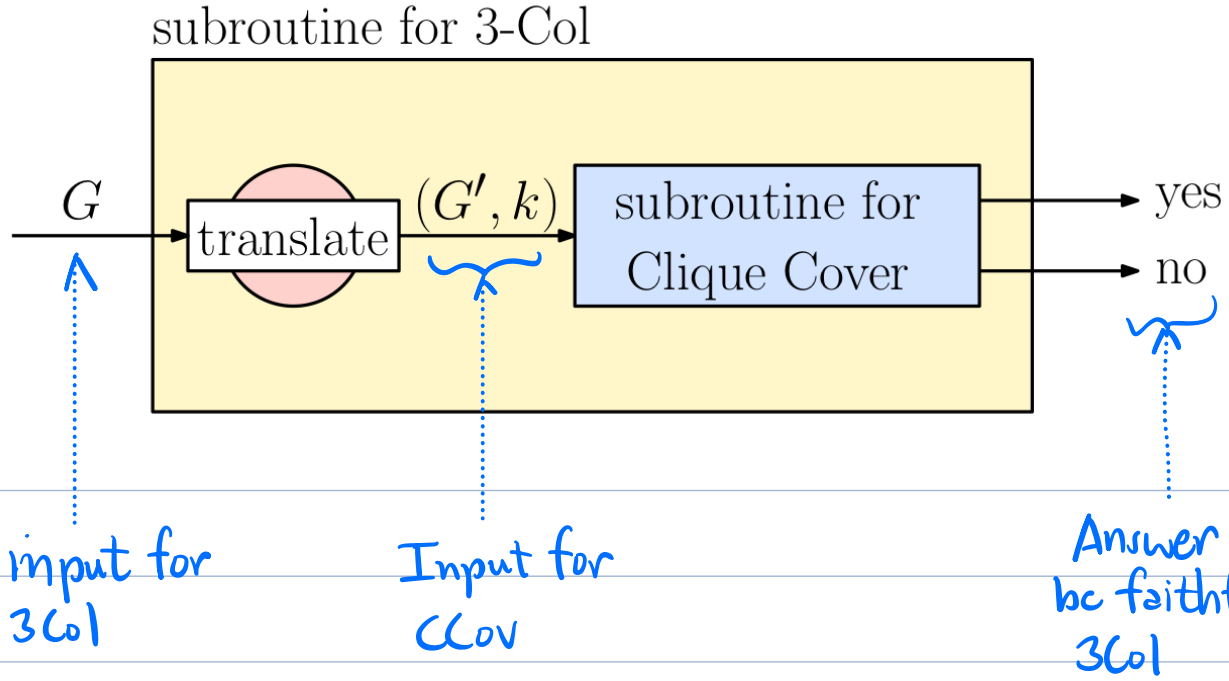
Want to show:

$$\text{CCOV} \in \text{P} \Rightarrow \text{3COL} \in \text{P}$$

If we could solve our problem efficiently — then — We could solve 3Col efficiently (No way!)

How? Suppose we had an efficient subroutine for CCov

Show we could use it to solve 3Col



How to translate?

- Both involve partioning vertices
  - 3Col - 3 color classes
  - CCov - k cliques

  } set k=3?
- Adjacency condition
  - 3Col - If u≠v have same color, $(u,v) \notin E$
  - CCov - If u≠v in same clique, $(u,v) \in E$

  } complement G!

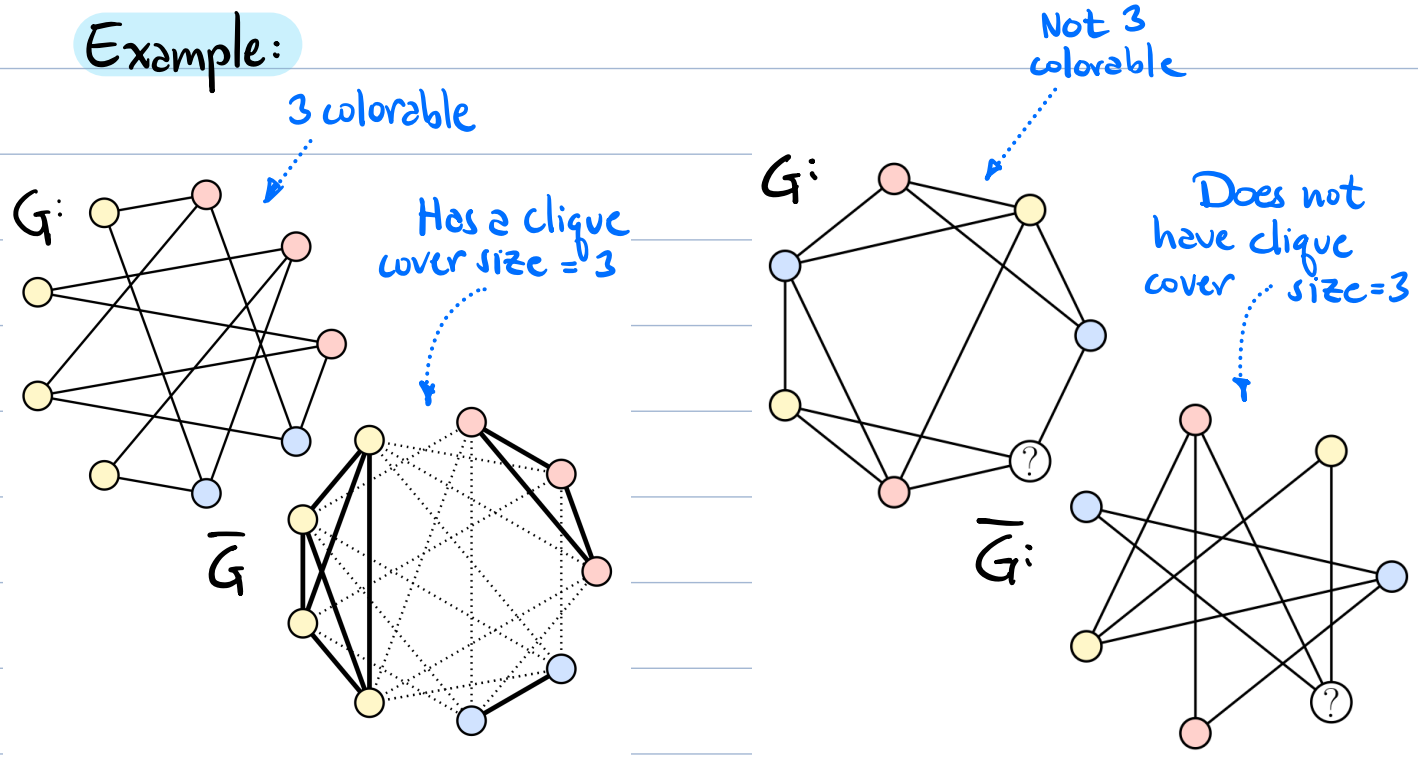
Reducing 3Col to CCov:

- Given G for 3Col:
  - Set $k \leftarrow 3$
  - Compute $\bar{G}$ (complement edge set)
- Run CCov on $(\bar{G}, k)$ + return whatever it returns

Is this a **faithful translation**?

> **Claim:** $G=(V,E)$ is 3-colorable iff $\bar{G}=(V,\bar{E})$ has a clique cover of size 3.

Example:



A **picture** is **not a proof**.

Proof:

($\Rightarrow$) If G is 3-colorable, let V_1, V_2, V_3 be partition of vertices into color classes.

If u & v in same color class, then $(u,v) \notin E$.

This implies that $(u,v) \in \bar{E}$ ← complement

$\Rightarrow$ All vertices of V_i (for $i=1,2,3$) are adjacent in $\bar{G}$

$\Rightarrow$ V_i is a clique in $\bar{G}$ (for $i=1,2,3$)

$\Rightarrow$ $\bar{G}$ has a clique cover of size 3. ✓

($\Leftarrow$) If $\bar{G}$ has a clique cover V_1, V_2, V_3 (complement → $\bar{\ }$)

then if u,v in same set V_i, then $(u,v) \in \bar{E}$

$\Rightarrow$ $(u,v) \notin E$

$\Rightarrow$ All vertices of V_i (for $i=1,2,3$) are non-adjacent in G

$\Rightarrow$ V_1, V_2, V_3 define a 3-coloring in G

$\Rightarrow$ G is 3-colorable ✓

Finally, note that our reduction runs in polynomial time (complement G)

- It did not try to color G
- It does not even know whether G can be 3-colored

Polynomial Reducibility:

Def: Language L_1 is **polynomial-time reducible** to language L_2 (denoted $L_1 \leq_p L_2$) if there is a **poly-time function f**, s.t.

$$\forall x \quad x \in L_1 \text{ iff } f(x) \in L_2$$

We showed $3Col \leq_p CCov$ (where $f(G) \to (\bar{G}, 3)$)

If $L_1 \leq_p L_2$ + L_2 is solvable in poly-time, so is L_1

also, if L_1 is not solvable in poly-time, neither is L_2

Summary:

Lemma: Given languages L_1, L_2, L_3

(i) $L_1 \leq_p L_2$ + $L_2 \in P$ $\Rightarrow L_1 \in P$

(ii) $L_1 \leq_p L_2$ + $L_1 \notin P$ $\Rightarrow L_2 \notin P$

(iii) $L_1 \leq_p L_2$ + $L_2 \leq_p L_3$ $\Rightarrow L_1 \leq_p L_3$

We now can define **NP-completeness.**

Intuitively- The "**hardest**" problems in NP.

Recall:

NP: Set of languages **verifiable in poly-time**

Def: A language L is **NP-hard** if

$$\forall\, L' \in NP,\ L' \leq_P L$$

Def: A language L is **NP-complete** if

(1) $L \in NP$ ⟵ verifiable

(2) L is NP-hard ⟵ but not easy!

This condition seems impossible, since it involves all languages in NP - an infinite set!

Next lecture: We'll prove there is an NP-complete problem

Summary:

- P - polynomial time
- NP - poly time verifiable - certificate
- Poly-time reducibility "$\leq_P$"
- NP-hardness
- NP-completeness