CMSC 451 - Algorithm Design
Lecture 16 - NP-Completeness: Reductions

Recap: So far we have discussed:

Decision Problems / Language Recognition:

All problems have yes/no outputs or equiv.

can be expessed as membership in a language

$MST = \{(G, z) \mid G \text{ has span. tree of wt.} \leq z\}$

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

Recall: Hamiltonian Cycle - Simple (non-repeating) cycle that visits all vertices

P: Class of languages (i.e., dec. problems) solvable in polynomial time.

$MST \in P$ (Run Kruskal + check weight $\leq z$)

NP: Class of languages L (ie., dec. problems) verifiable in polynomial time. That is, if

$x \in L$, there is a certificate

that allows us to prove $x \in L$ in poly. time

(e.g. if $G \in HC$, certificate is sequence of vertices. We can check it is a cycle + hits all vertices.)

NP = "Nondeterministic Poly. time"

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!

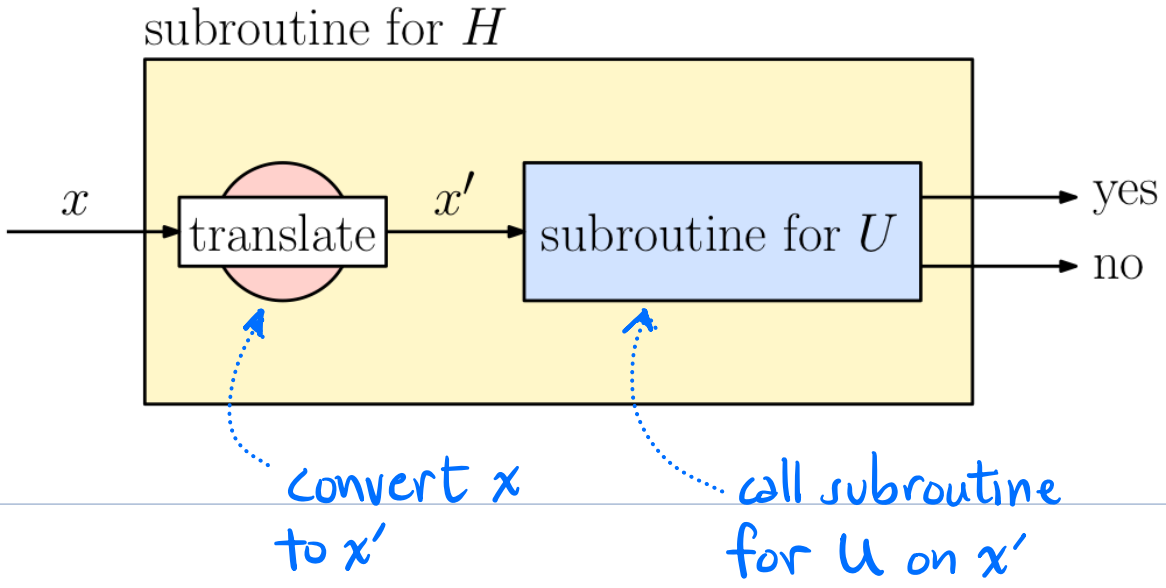
$U \in P \Rightarrow H \in P$

$U \in P$: 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

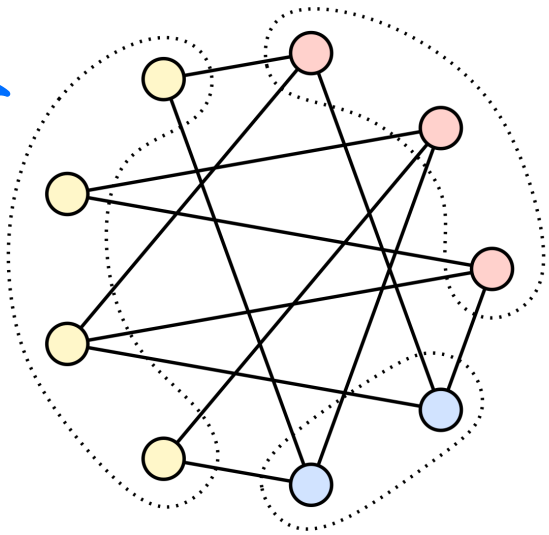
Pictorially: Is $x \in H$?



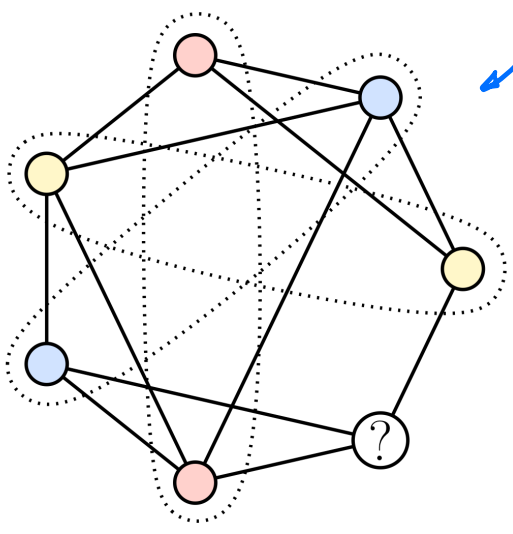
Example: Let's try this on two actual problems.

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



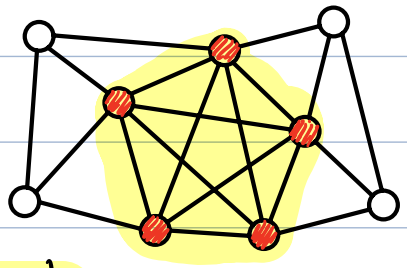
Not 3 colorable (Need at least 4)



No know poly-time algorithm - even for planar graphs

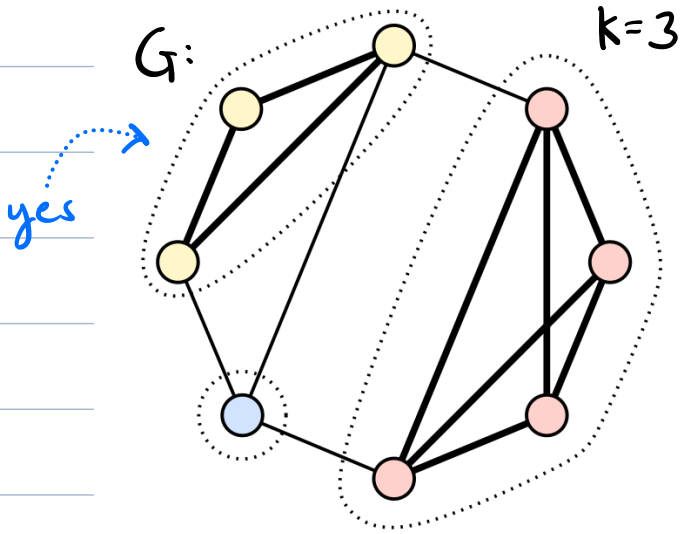
Your Problem (which you want to show is hard)

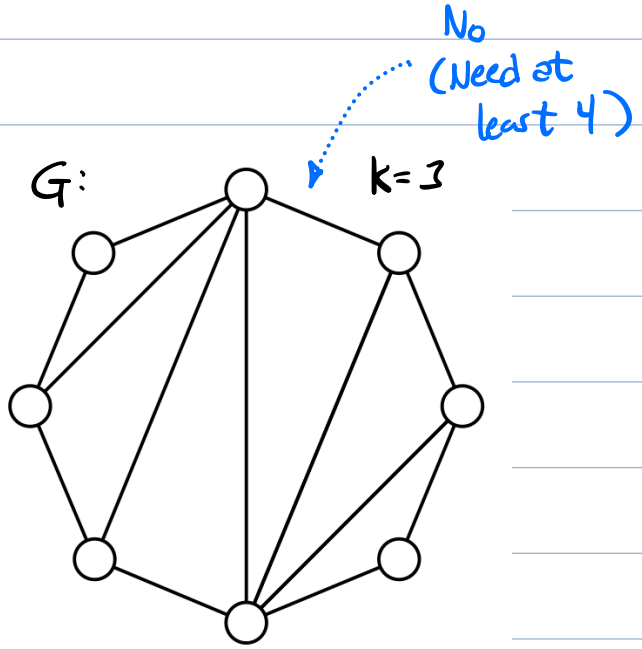
Def: A clique is a completely connected subgraph



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:





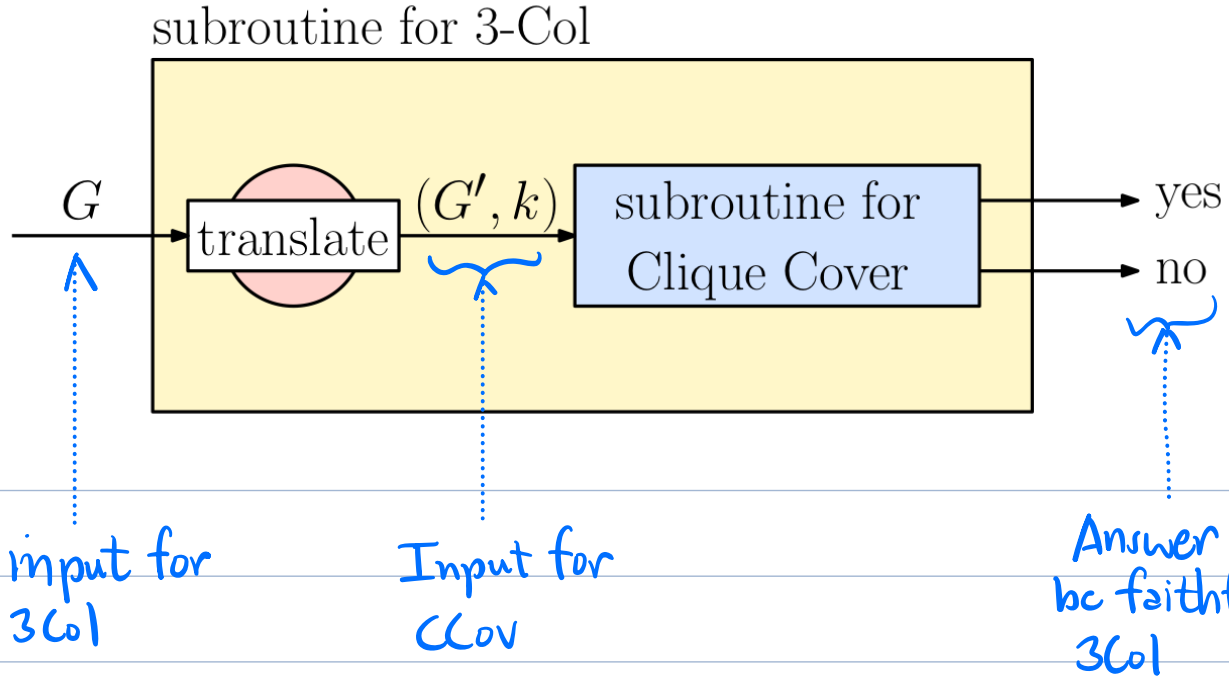
Want to show:

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

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!

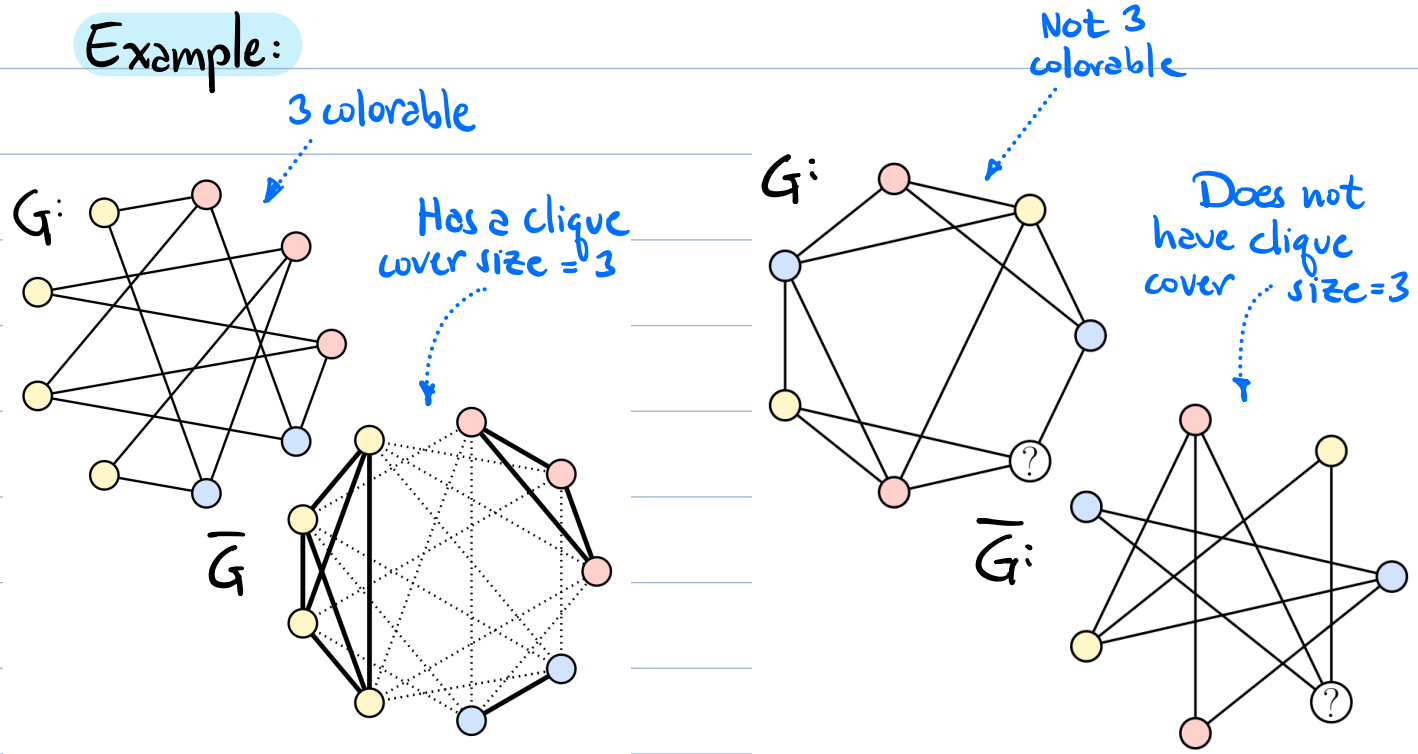
Reducing 3Col to CCov:

- 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!

An equivalent (+ more workable) definition.

Def: A language L is NP-complete if

(1) $L \in NP$

(2) $L' \leq_p L$, where L' is known to be NP-hard

But, for this to apply, we need an initial NP-hard language.

Next lecture:

SAT - language of boolean formulas that are satifiable

Cook's Theorem: SAT is NP-complete

Summary:

(1) Poly-time reducibility "$\leq_P$"

(2) NP-hardness

(3) NP-completeness