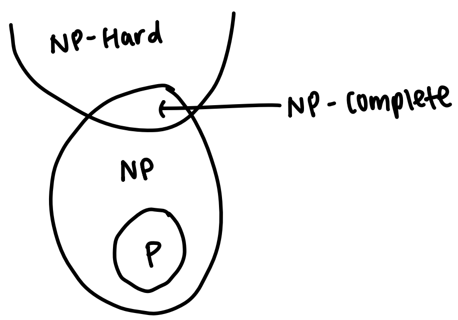
Complexity Classes:

P (polynomial): solvable in polynomial time

NP (nondeterministic polynomial): solution can be verified in polynomial time

NP-Hard: all problems in NP can reduce to an NP-hard problem

NP-Complete: a problem in NP and NP-Hard



$P \stackrel{?}{=} NP$

Reductions

$A \longrightarrow B$

If A reduces to B in polynomial time, it means an algorithm for B can be used to solve A. B is at least as hard as A.

A $\longrightarrow$ preprocessing (poly time) $\longrightarrow$ $\boxed{\text{algo for B}}$ $\longrightarrow$ post processing (poly time) $\longrightarrow$ solution to A

Must prove that:

1. Solution in A $\longrightarrow$ Solution in B
   Reduction incomplete if A has a solution but the algorithm for B can't find it.
2. Solution in B $\longrightarrow$ Solution in A
   Reduction incorrect if algorithm for B finds a solution that does not map to an actual solution in A.

Showing NP-Completeness

To show a problem A is NP-Complete:

1. Show there is a polynomial verifier ($A \in NP$)
2. Reduce another NP-Complete problem to A ($A \in$ NP-Hard)

1 NP or not NP, that is the question

For the following questions, circle the (unique) condition that would make the statement true.

(a) If B is **NP**-complete, then for any problem $A \in$ **NP**, there exists a polynomial-time reduction from A to B.

$A \to B$

Always True   True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

by definition

(b) If B is in **NP**, then for any problem $A \in$ **P**, there exists a polynomial-time reduction from A to B.

reduction can be just solving A

Always True   True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

(c) 2 SAT is **NP**-complete.

2SAT $\in$ P

if P = NP, 3SAT $\to$ 2SAT

Always True   True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

(d) Minimum Spanning Tree is in **NP**.

Always True   True iff $\mathbf{P} = \mathbf{NP}$ True iff $\mathbf{P} \neq \mathbf{NP}$ Always False

P $\subseteq$ NP

can be checked in polynomial time

2 California Cycle

Prove that the following problem is NP-hard

Input: A directed graph $G = (V, E)$ with each vertex colored blue or gold, i.e., $V = V_{\text{blue}} \cup V_{\text{gold}}$

Goal: Find a *Californian cycle* which is a directed cycle through all vertices in G that alternates between blue and gold vertices (Hint : Directed Rudrata Cycle)

To prove NP-hard : Directed Rudrata Cycle $\to$ California Cycle

Use California cycle to find rudrata cycle.

Rudrata Cycle / Hamiltonian cycle: cycle in graph that starts and ends at vertex v and visits all other vertices exactly once

Reduction:

Given $G = (V, E)$, construct new graph $G' = (V', E')$

For each $v \in V$, create blue node v_b with edge to gold node v_g

• becomes • $\to$ •

For each edge $(u, v) \in E$, add edge (u_g, v_b) to E'

$u \to v$ becomes $u_b \to u_g \to v_b \to v_g$

proof:

1. rudrata cycle in G $\to$ california cycle in G'
   for each edge $(u,v) \in G$ that is in rudrata cycle, follow the path $u_b \to u_g \to v_b \to v_g$ in G'
   visit all vertices ✓ alternate blue/gold ✓
2. california cycle in G' $\to$ rudrata cycle in G
   Each path $u_b \to u_g \to v_b \to v_g$ in G' equivalent to $u \to v$ in G.
   visit all vertices exactly once ✓

3 NP Basics

$A \to B$ B at least as hard as A

Assume A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. What information can you derive of the other problem given each fact? Each part should be considered independent; i.e., you should not use the fact given in part (a) as part of your analysis of part (b).

1. A is in **P**. Nothing
2. B is in **P**. A in P
3. A is **NP**-hard. B is NP-Hard
4. B is **NP**-hard. Nothing

4 Local Search for Max Cut

Attendance: tinyurl.com/disc10cs170

Sometimes, local search algorithms can give good approximations to NP-hard problems. In the Max-Cut problem, we have an unweighted graph $G(V, E)$ and we want to find a cut (S, T) with as many edges "crossing" the cut (i.e. with one endpoint in each of S, T) as possible. One local search algorithm is as follows: Start with any cut, and while there is some vertex $v \in S$ such that more edges cross $(S - v, T + v)$ than (S, T) (or some $v \in T$ such that more edges cross $(S + v, T - v)$ than (S, T)), move v to the other side of the cut. Note that when we move v from S to T, v must have more neighbors in S than T.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

$|E|$

- each move must increase edges crossing the cut by at least 1
- cut size between 0 and $|E|$

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

$\delta_{in}(v)$: # edges from v to vertices on same side of cut

$\delta_{out}(v)$: # edges from v to vertices on other side of cut

$\delta_{out}(v) \geq \delta_{in}(v)$

Total edges crossing cut: $\frac{1}{2} \sum_{v \in V} \delta_{out}(v)$

Total edges in graph: $\frac{1}{2} \sum_{v \in V} \delta_{in}(v) + \delta_{out}(v)$

$$\frac{1}{2}|E| = \frac{1}{4} \sum_{v \in V} \delta_{in}(v) + \delta_{out}(v) \leq \frac{1}{4} \sum_{v \in V} \delta_{out}(v) + \delta_{out}(v) = \frac{1}{2} \sum_{v \in V} \delta_{out}(v)$$

5 Cycle Cover

In the cycle cover problem, we have a directed graph G, and our goal is to find a set of directed cycles $C_1, C_2, \ldots C_k$ in G such that every vertex appears in exactly one cycle (a cycle cannot revisit vertices, e.g. $a \to b \to a \to c \to a$ is not a valid cycle, but $a \to b \to c \to a$ is), or declare none exists.

In the bipartite perfect matching problem, we have a undirected bipartite graph (a graph where the vertices can be split into L, R, and there are no edges between two vertices in L or two vertices in R), and our goal is to find a set of edges in this graph such that every vertex is adjacent to exactly one edge in the set, or declare none exists.

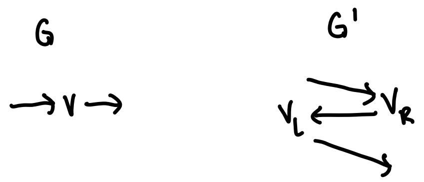
Give a reduction from cycle cover to bipartite perfect matching. (Hint: In a cycle cover, every vertex has one incoming and one outgoing edge.)

cycle cover $\rightarrow$ bipartite perfect matching

Given cycle cover graph G, create bipartite graph G'

For every vertex v in G, create vertices v_L and v_R. Add edge (v_L, v_R) to G'.

G has cycle cover $\rightarrow$ G' has perfect matching



G' has perfect matching $\longrightarrow$ G has cycle cover

If edges (a_L, b_R) $(b_L, c_R) \ldots (z_L, a_R)$ in G'

$a \to b \to c \to \ldots \to z \to a$ in G

since v_L and v_R are both adjacent to some edge, every vertex included in cycle cover