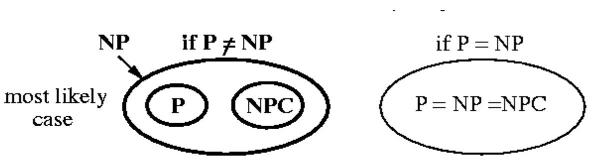
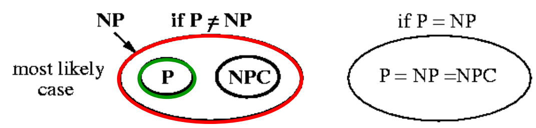
Unit 9: NP-Completeness

- Course contents:
  - Complexity classes
  - Reducibility and NP-completeness proofs
- Readings:
  - Chapter 34



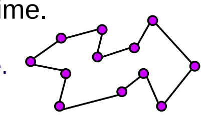
Complexity Classes

- Developed by S. Cook and R. Karp in early 1970.
- **The class P:** class of problems that can be **solved** in polynomial time in the **size of input**.
  - **Size of input:** size of encoded "binary" strings.
  - Edmonds: Problems in P are considered **tractable**.
  - Closed under addition, multiplication, composition, **complement**, etc.
- **The class NP** (**N**ondeterministic **P**olynomial): class of problems that can be **verified** in polynomial time in the size of input.
  - P = NP?
- **The class NP-complete (NPC): Any** NPC problem can be solved in polynomial time ⇒ **All** problems in NP can be solved in polynomial time.



Verification Algorithm and Class NP

- **Verification algorithm:** a 2-argument algorithm A, where one argument is an input string x and the other is a binary string y (called a **certificate**). A verifies x if there exists y s.t. A answers "yes."
- Exp: **The Traveling Salesman Problem (TSP)**
  - **Instance:** a set of n cities, distance between each pair of cities, and a bound B.
  - **Question:** is there a route that starts and ends at a given city, visits every city exactly once, and has total distance $\leq B$?
- Is TSP $\in$ NP?
- Need to **check** a solution in polynomial time.
  - Guess a tour (certificate).
  - Check if the tour visits every city exactly once.
  - Check if the tour returns to the start.
  - Check if total distance $\leq B$.



- All can be done in $O(n)$ time, so TSP $\in$ NP.

Decision & Optimization Problems

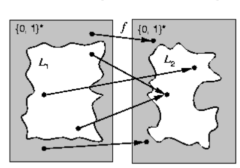
- **Decision problems:** those having yes/no answers.
  - MST: Given a graph $G=(V, E)$ and a bound K, **is there a spanning tree with a cost at most K?**
  - TSP: Given a set of cities, distance between each pair of cities, and a bound B, **is there a route that** starts and ends at a given city, visits every city exactly once, and **has total distance at most B?**
- **Optimization problems:** those finding a legal configuration such that its cost is minimum (or maximum).
  - MST: Given a graph $G=(V, E)$, **find the cost of a minimum spanning tree** of G.
  - TSP: Given a set of cities and the distance between each pair of cities, **find the distance of a "minimum route"** starts and ends at a given city and visits every city exactly once.

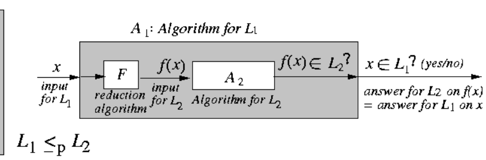
Decision vs. Optimization Problems

- Could apply binary search on a decision problem to obtain solutions to its optimization problem.
- **NP-completeness is associated with decision problems.**
- c.f., **Optimal** solutions/costs, optimal (**exact**) algorithms
  - optimal ≠ exact in the theoretic computer science community.

Polynomial-time Reduction

- **Motivation:** Let L_1 and L_2 be two decision problems. Suppose algorithm A_2 can solve L_2. Can we use A_2 to solve L_1?
  - E.g., maximum cardinality bipartite matching (L_1) vs. maximum flow (L_2)
- **Polynomial-time reduction f from L_1 to L_2: $L_1 \leq_P L_2$**
  - f reduces input for L_1 into an input for L_2 s.t. the reduced input is a "yes" input for L_2 iff the original input is a "yes" input for L_1.
    - $L_1 \leq_P L_2$ if $\exists$ polynomial-time computable function f: $\{0, 1\}^* \rightarrow \{0, 1\}^*$ s.t. $x \in L_1$ **iff** $f(x) \in L_2$, $\forall x \in \{0, 1\}^*$.
    - **L_2 is at least as hard as L_1.**
- f is computable in polynomial time.

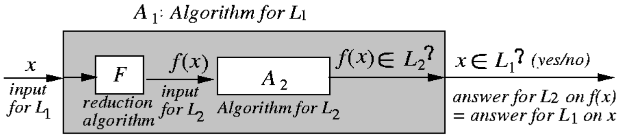




$L_1 \leq_P L_2$

Significance of Reduction

- Significance of $L_1 \leq_P L_2$:
  - ∃ polynomial-time algorithm for $L_2 \Rightarrow$ ∃ polynomial-time algorithm for L_1 ($L_2 \in$ P $\Rightarrow L_1 \in$ P).
  - ∄ polynomial-time algorithm for $L_1 \Rightarrow$ ∄ polynomial-time algorithm for L_2 ($L_1 \notin$ P $\Rightarrow L_2 \notin$ P).
- $\leq_P$ is transitive, i.e., $L_1 \leq_P L_2$ and $L_2 \leq_P L_3 \Rightarrow L_1 \leq_P L_3$.

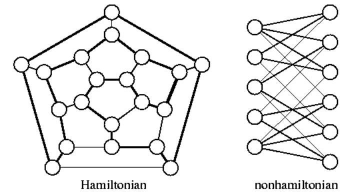


L_1: system of difference constraint vs. L_2: SSSP

L_1: Bipartite cardinality matching vs. L_2: maximum flow

Polynomial Reduction: HC $\leq_P$ TSP

- **The Hamiltonian Circuit Problem (HC)**
  - **Instance:** an undirected graph $G = (V, E)$.
  - **Question:** is there a cycle in G that includes every vertex exactly once?
- **TSP: The Traveling Salesman Problem**
- **Claim:** HC $\leq_P$ TSP.
  1. Define a function f mapping **any** HC instance into a TSP instance, and show that ***f* can be computed in polynomial time**.
  2. Prove that G has an HC iff the reduced instance has a TSP tour **with distance ≤ *B*** ($x \in$ HC ⇔ $f(x) \in$ TSP).



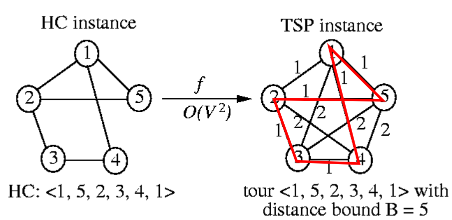
HC $\leq_P$ TSP: Step 1

1. Define a reduction function f for HC $\leq_P$ TSP.
   - Given an HC instance $G = (V, E)$ with n vertices
     - Create a set of n cities labeled with names in V.
     - Assign distance between u and v

$$d(u,v) = \begin{cases} 1, & if\ (u,v) \in E, \\ 2, & if\ (u,v) \notin E. \end{cases}$$

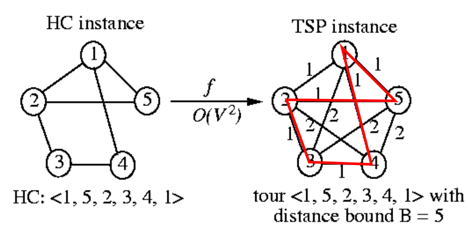
     - Set bound $B = n$.
   - f can be computed in $O(V^2)$ time.

look for the difference between the two problems to make the reduction!!



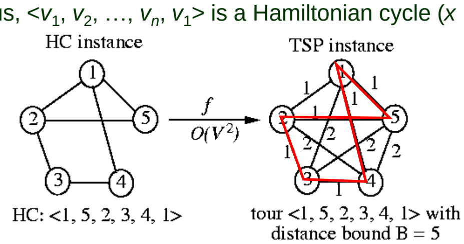
HC $\leq_P$ TSP: Step 2

2. G has an HC iff the reduced instance has a TSP **with distance ≤ *B***.
   - $x \in$ HC $\Rightarrow f(x) \in$ TSP.
     - Suppose the HC is $h = <v_1, v_2, \ldots, v_n, v_1>$. Then, h is also a tour in the transformed TSP instance.
     - The distance of the tour h is $n = B$ since there are n consecutive edges in E, and so has distance 1 in $f(x)$.
     - Thus, $f(x) \in$ TSP ($f(x)$ has a TSP tour with distance $\leq B$).



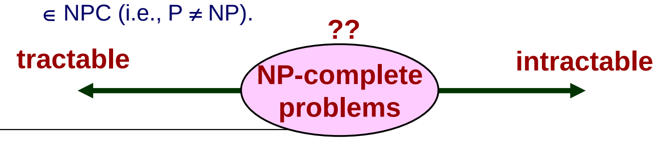
HC $\leq_P$ TSP: Step 2 (cont'd)

2. G has an HC iff the reduced instance has a TSP **with distance ≤ *B***.
   - $f(x) \in$ TSP $\Rightarrow x \in$ HC.
     - Suppose there is a TSP tour **with distance ≤ *n* = *B***. Let it be $<v_1, v_2, \ldots, v_n, v_1>$..
     - Since **distance of the tour ≤ *n*** and **there are *n* edges** in the TSP tour, the tour contains only edges in E since all edge weights are equal to 1.
     - Thus, $<v_1, v_2, \ldots, v_n, v_1>$ is a Hamiltonian cycle ($x \in$ HC).



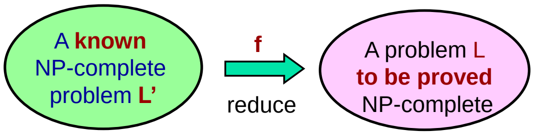
NP-Completeness

- A **decision** problem L (a language $L \subseteq \{0, 1\}^*$) is **NP-complete (NPC)** if
  1. $L \in$ NP, and
  2. $L' \leq_P L$ for every $L' \in$ NP.
- **NP-hard:** If L satisfies property 2, but not necessarily property 1, we say that L is **NP-hard**.
- Suppose $L \in$ NPC.
  - If $L \in P$, then there exists a polynomial-time algorithm for every $L' \in$ NP (i.e., P = NP).
  - If $L \notin P$, then there exists no polynomial-time algorithm for any $L' \in$ NPC (i.e., P ≠ NP).



Proving NP-Completeness

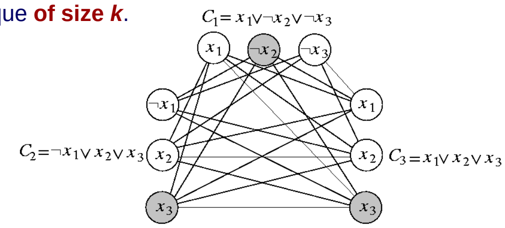
- **Five steps for proving that *L* is NP-complete:**
  1. Prove $L \in$ NP.
  2. Select a known NP-complete problem ***L'***.
  3. Construct a reduction *f* transforming **every** instance of ***L'*** to an instance of *L*.
  4. Prove that $x \in L'$ iff $f(x) \in L$ for all $x \in \{0, 1\}^*$.
  5. Prove that *f* is a polynomial-time transformation.



Here we intend to show how difficult L is!!
Cf. matching $\le_P$ maximum flow to show how easy matching is!!

Clique is NP-Complete

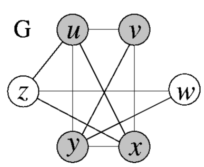
- A **clique** in $G = (V, E)$ is a complete subgraph of G.
- **The Clique Problem (Clique)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \le |V|$.
  - **Question:** is there a clique $V' \subseteq V$ of size $\ge k$?
- **Clique $\in$ NP.**
- **Clique is NP-hard:** 3SAT $\le_P$ Clique.
  - **Key:** Construct a graph G such that ϕ is satisfiable $\Leftrightarrow$ G has a clique **of size *k***.



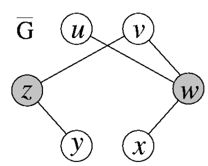
satisfying assignment: <x1, x2, x3> = <x, 0, 1>

Vertex-Cover is NP-Complete

- A **vertex cover** of $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(w, v) \in E$, then $w \in V'$ or $v \in V'$.
- **The Vertex-Cover Problem (Vertex-Cover)**
  - **Instance:** a graph $G = (V, E)$ and a positive integer $k \le |V|$.
  - **Question:** is there a subset $V' \subseteq V$ of size $\le k$ such that each edge in E has at least one vertex (endpoint) in V'?
- **Vertex-Cover $\in$ NP.**
- **Vertex-Cover is NP-hard:** Clique $\le_P$ Vertex-Cover.
  - **Key: complement** of G: $\bar{G} = (V, \bar{E})$, $\bar{E} = \{(w, v): (w, v) \notin E\}$.



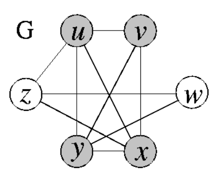
clique V' = {u, v, x, y}



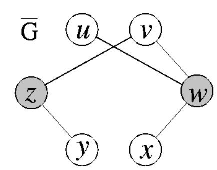
vertex cover V – V' = {w, z}

Vertex-Cover is NP-Complete (cont'd)

- G Has a Clique of Size $k \Rightarrow \bar{G}$ Has a Vertex Cover of size $|V| - k$.
  - Suppose that G has a clique $V' \subseteq V$ with $|V'| = k$.
  - Let (w, v) be any edge in $\bar{E} \Rightarrow (w, v) \notin E \Rightarrow$ at least one of w or v does not belong to V'
  - So, $w \in V - V'$ or $v \in V - V' \Rightarrow$ edge (w, v) is covered by $V - V'$.
  - Thus, $V - V'$ forms a vertex cover of $\bar{G}$, and $|V - V'| = |V| - k$.



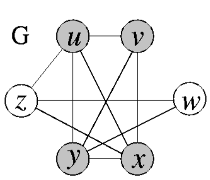
clique V' = {u, v, x, y}



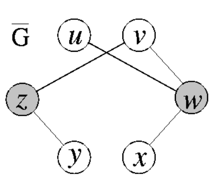
vertex cover V – V' = {w, z}

Vertex-Cover is NP-Complete (cont'd)

- $\bar{G}$ Has a Vertex Cover of size $|V| - k \Rightarrow G$ Has a Clique of Size k.
  - Suppose that $\bar{G}$ has a vertex cover $V' \subseteq V$ with $|V'| = |V| - k$.
  - $\forall a, b \in V$, if $(a, b) \in \bar{E}$, then $a \in V'$ or $b \in V'$ or both.
  - So, $\forall a, b \in V$, if $a \notin V'$ and $b \notin V'$, $(a, b) \in E \Rightarrow V - V'$ is a clique, and $|V| - |V'| = k$.



clique V' = {u, v, x, y}



vertex cover V – V' = {w, z}