Algorithm Analysis and Data Structures

CSCI 7432 - Fall 2022

NP-Related Concepts

Dr. Yao XU

Assistant Professor

Department of Computer Science

Georgia Southern University

Email: yxu@georgiasouthern.edu

Table of Contents

Complexity Classes: P and NP

Complexity Classes

- ***Computational complexity***: the amount of computational resources required to solve a given task.
- A problem with input size n can be solved by a *polynomial-time* algorithm if its worst-case running time is $O(n^k)$ for some constant k.
  - Problems that are solvable by *polynomial-time* algorithms are called ***tractable***, or *easy*;
  - Problems that require *superpolynomial time* ($\omega(n^k)$ for some constant k) are called ***intractable***, or *hard*.
- Three classes of problems:
  - **P** – Problems that are solvable in polynomial time
  - **NP** – Problems that are "***verifiable***" in polynomial time
  - **NPC** (***NP-Complete***) – The "hardest" problems in **NP**

    Problems in **NPC** are intractable (if **P** ≠ **NP**)

Decision Problems

- The class **P** – ***Decision problems*** that are solvable in polynomial time.
- ***Decision problems***: Problems that can be answered by "yes" or "no".
- An optimization problem can be casted as a related decision problem.

  Example: SHORTEST-PATH

  - Optimization problem: Given an undirected graph G and two vertices u and v, find a path from u to v that uses the fewest number of edges.
  - Decision problem: Given an undirected graph G, two vertices u and v, and an integer k, does a path exist from u to v consisting of at most k edges?
- A decision problem is in a sense "easier," or at least "no harder", than its related optimization problem.

The Class NP

- The class **NP** – **Decision problems** whose solutions are ***verifiable*** in polynomial time.
  - That is, given an instance I and a solution S, we can verify that S is a solution to I in polynomial time (polynomial in $|I|$).
- **NP** stands for "***nondeterministic*** polynomial time".
  - The class **NP** – Decision problems that are solvable in polynomial time by a ***nondeterministic Turing machine***.
- Relationship between **P** and **NP**
  - **P** $\subseteq$ **NP**: A decision problem that is solvable in polynomial time must also be verifiable in polynomial time.
  - **P** $=$ **NP**? - Unknown! (An open problem since 1970s)

    It is widely believed that **P** $\neq$ **NP**.

Examples (1/2)

Example 1: (**SP**, decision version of the Shortest Path problem) Given an undirected graph G, two vertices u and v, and an integer k, does a path exist from u to v consisting of at most k edges?

1. **SP** $\in$ **NP** ?

   Given $I = \langle G, u, v, k \rangle$ and P, can we verify P is a solution to I in polynomial time?

   - Check if P is a path from u to v
   - Check if the number of edges on path P is $\leq k$

   A: **SP** $\in$ **NP**

2. **SP** $\in$ **P** ?

   - Find shortest path from u to every other vertex – Run BFS
   - Check if $v.d \leq k$

   A: **ST** $\in$ **P**

Examples (2/2)

Example 2: (**IntK**, decision version of the Integral Knapsack problem) Given n items, with integer weight w_i and integer value v_i for each item, a knapsack with capacity W, and an integer V, is there a subset of items with total weight at most W and total value at least V?

1. **IntK** $\in$ **NP** ?

   Given $I = \langle n, w, v, W, V \rangle$ and S, can we verify S is a solution to I in polynomial time?

   - Check if total weight $\leq W$ and total value $\geq V$

   A: **IntK** $\in$ **NP**

2. **IntK** $\in$ **P** ?

   - DP for Integral Knapsack takes $\Theta(nW)$ time - *pseudo-polynomial*

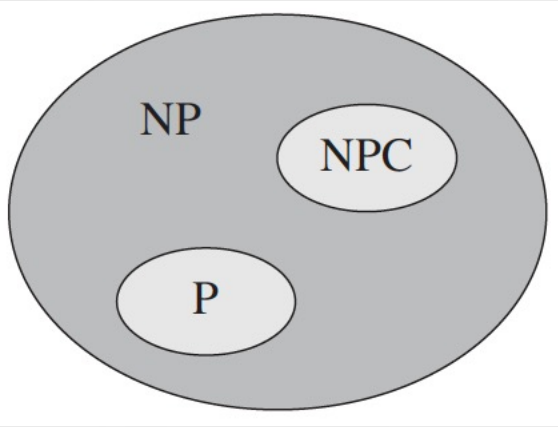
   A: Not known.

Complexity Class: NPC

NP-Completeness and Reducibility

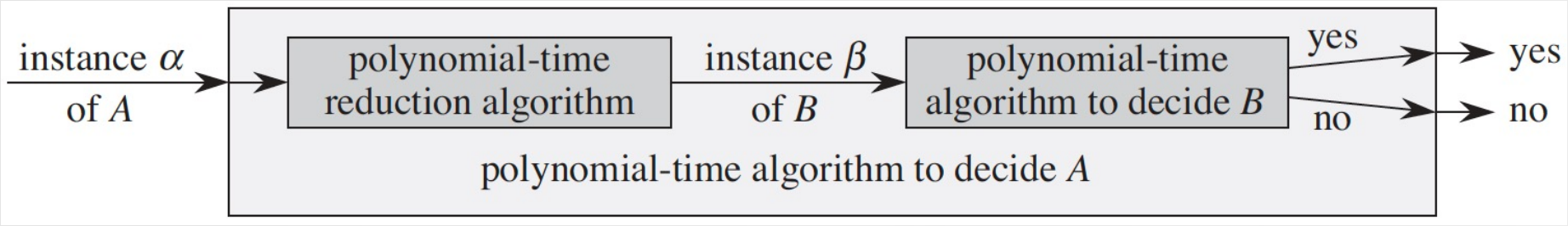
NP-Completeness

- If **P** ≠ **NP**, problems in **NPC** are intractable.
- That is, if **P** ≠ **NP**, then all ***NP-complete*** problems are not in **P**.
  - *Contrapositive:* If an ***NP-complete*** problem can be solved in polynomial time, then all problems in **NP** can be solved in polynomial time.
- A decision problem A is ***NP-complete*** if
  1. A is in **NP**, and
  2. Every problem in **NP** is ***polynomial-time reducible*** to A.
- Relationships among **P**, **NP**, and **NPC** (if **P** ≠ **NP**):



Reducibility

- A decision problem A is said to be ***polynomial-time reducible*** to a decision problem B, written $A \leq_P B$, if there is a procedure t that can transform any instance α of problem A to an instance β of problem B such that
  - t runs in polynomial-time (t is called a polynomial-time ***reduction algorithm***);
  - The answer for α is "yes" iff the answer for β is also "yes".



- If we can solve problem B in polynomial time, then we can use that algorithm to solve problem A in polynomial time.
  - *Contrapositive:* If problem A is not in **P**, then problem B is not in **P**.

Complexity Class: NPC

NP-Completeness Proofs

A First NP-Complete Problem

- The ***circuit-satisfiability*** problem
  - *Problem definition can be found on p.1070-1072 of the textbook*
  - NP-completeness proof: *Theorem 34.7 on p.1077 of the textbook*
- The ***(formula) satisfiability (SAT)*** problem
  - Problem definition on next slide
  - NP-completeness proof:
    - *Theorem 34.9 on p.1080 of the textbook* (prove by reduction from circuit-satisfiability)
    - ***Cook-Levin Theorem*** (***Cook's Theorem***), which shows that every problem in **NP** is polynomial-time reducible to **SAT** (the ***CNF-SAT*** problem, to be specific).

The Satisfiability (SAT) Problem

- Notions for a Boolean formula ϕ:
  - ***Boolean variables***: $x_1, x_2, \cdots, x_n$
  - ***Boolean operations***: AND ($\wedge$), OR ($\vee$), NOT ($\neg$), implication ($\rightarrow$), iff ($\leftrightarrow$)
  - ***Truth assignment***: a set of values for all variables of ϕ
  - ***Satisfying (truth) assignment***: a truth assignment that causes the Boolean formula ϕ to evaluate to true
- The ***SAT*** problem: Given a Boolean formula ϕ, is ϕ satisfiable?
- **Example:** $\phi = \Big((x_1 \rightarrow x_2) \vee \neg\big((\neg x_1 \leftrightarrow x_3) \vee x_4\big)\Big) \wedge \neg x_2$
  - **Q:** Is ϕ satisfiable?
  - **A:** Yes. ϕ has the satisfying truth assignment:

$$x_1 = \mathrm{F}, x_2 = \mathrm{F}, x_3 = \mathrm{T}, x_4 = \mathrm{T}.$$

The CNF-SAT Problem

- Additional notions:
  - ***Literal***: x or $\neg x$
  - ***Clause***: disjunction of literals, e.g., $x_1 \vee x_2 \vee \neg x_3$
  - ***Conjunctive normal form (CNF)***: conjunction of clauses, e.g.,

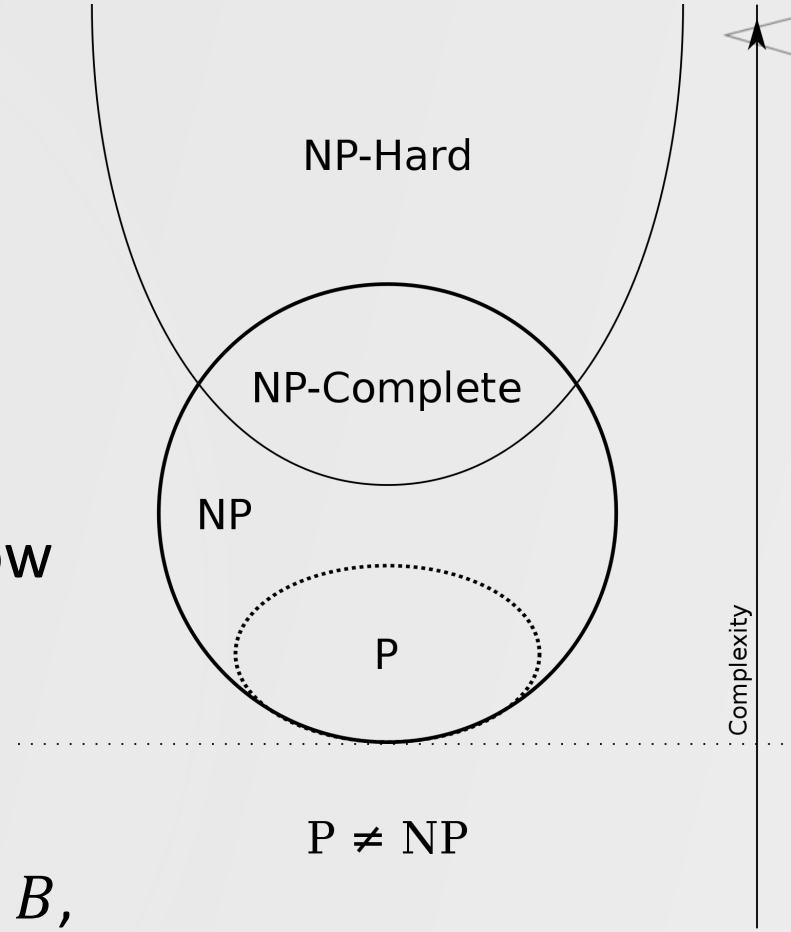
$$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3).$$

- The ***CNF-SAT*** problem: Given a Boolean CNF formula ϕ, is ϕ satisfiable?
- **Example:** $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$
  - **Q:** Is ϕ satisfiable?
  - **A:** Yes. ϕ has the satisfying truth assignment:

$$x_1 = \mathrm{T}, x_2 = \mathrm{T}, x_3 = \mathrm{F}.$$

NP-Completeness Proof

- *Recall:* A decision problem A is ***NP-complete*** if
  1. A is in **NP**, and
  2. Every problem in **NP** is polynomial-time reducible to A.
- *Note:* If a problem satisfies condition 2, but not necessarily condition 1, then we say it is ***NP-hard***.
- To show that a problem B is NP-complete, we show
  1. B is in **NP**
  2. B is **NP-hard**
     1) Select a known NP-complete problem A;
     2) Show that A is polynomial-time reducible to B, i.e., $A \leq_P B$.



NP-Completeness Proof of 3-CNF-SAT (1/3)

- *Recall the **CNF-SAT** problem*: Given a Boolean CNF formula ϕ, is ϕ satisfiable?
- ***3-CNF-SAT*** problem: A CNF-SAT problem with every clause containing exactly three distinct literals. E.g.,

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4).$$

NP-Completeness Proof:

1. **Show: 3-CNF-SAT is in NP**
   - Given a Boolean 3-CNF formula ϕ with n variables, m clauses, and a truth assignment for $x_1, x_2, \cdots, x_n$
   - For each clause, it takes $O(1)$ time to check if at least one literal is T.

   Can be verified in $O(m)$ time.

NP-Completeness Proof of 3-CNF-SAT (2/3)

NP-Completeness Proof (cont'd):

2. Show: CNF-SAT $\leq_P$ 3-CNF-SAT

1) For any CNF-SAT formula ϕ, construct a 3-CNF-SAT formula ϕ' in $O(n)$ time.

- For 1-literal clause (x_i), construct a 3-CNF formula:
  $(x_i \vee y_1 \vee y_2) \wedge (x_i \vee y_1 \vee \neg y_2) \wedge (x_i \vee \neg y_1 \vee y_2) \wedge (x_i \vee \neg y_1 \vee \neg y_2)$.
- For 2-literal clause $(x_i \vee x_j)$, construct a 3-CNF formula:
  $(x_i \vee x_j \vee y_1) \wedge (x_i \vee x_j \vee \neg y_1)$.
- For 3-literal clause $(x_i \vee x_j \vee x_k)$, unchanged.
- For more-than-3-literal clause $(x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k})$, construct a 3-CNF formula:
  $(x_{i_1} \vee x_{i_2} \vee y_1) \wedge (\neg y_1 \vee x_{i_3} \vee y_2) \wedge \cdots \wedge (\neg y_{k-3} \vee x_{i_{k-1}} \vee x_{i_k})$.
- All y literals are distinct for different clauses.

NP-Completeness Proof of 3-CNF-SAT (3/3)

NP-Completeness Proof (cont'd):

2. Show: CNF-SAT $\leq_P$ 3-CNF-SAT

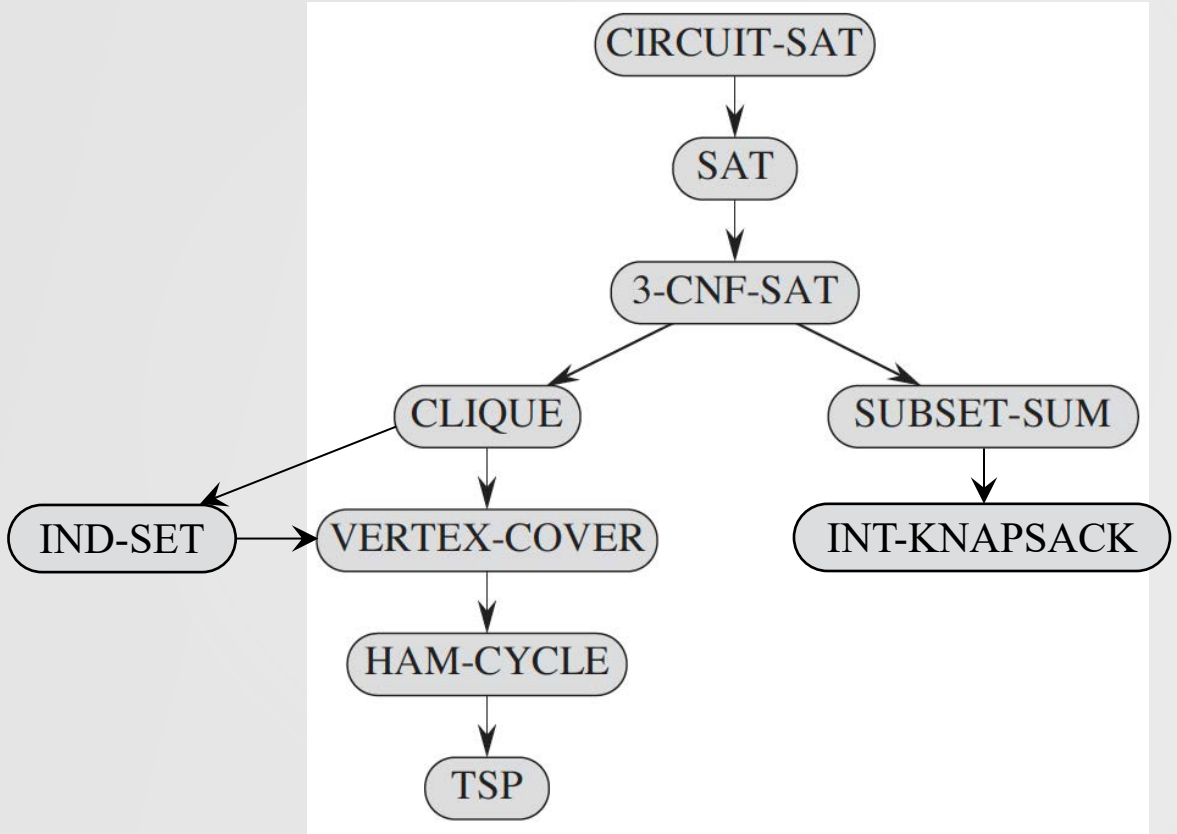
2) Formula ϕ is satisfiable iff formula ϕ' is satisfiable.

- For 1-literal clause in ϕ,
  $(x_i) = \text{T}$ iff in ϕ', $(x_i \vee y_1 \vee y_2) \wedge (x_i \vee y_1 \vee \neg y_2) \wedge (x_i \vee \neg y_1 \vee y_2) \wedge (x_i \vee \neg y_1 \vee \neg y_2) = \text{T}$.
- For 2-literal clause in ϕ,
  $(x_i \vee x_j) = \text{T}$ iff in ϕ', $(x_i \vee x_j \vee y_1) \wedge (x_i \vee x_j \vee \neg y_1) = \text{T}$.
- For more-than-3-literal clause in ϕ,
  $(x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k})$ is satisfiable iff in ϕ',
  $(x_{i_1} \vee x_{i_2} \vee y_1) \wedge (\neg y_1 \vee x_{i_3} \vee y_2) \wedge \cdots \wedge (\neg y_{k-3} \vee x_{i_{k-1}} \vee x_{i_k})$ is satisfiable.

NP-Complete Problems

NP-Complete Problems

- Each problem in the figure can be proved to be NP-complete by a polynomial-time reduction from the problem that points to it.



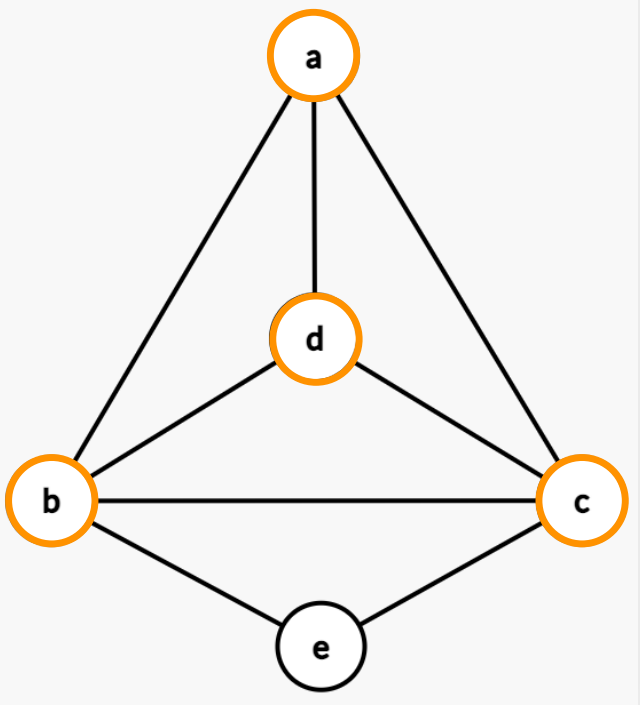
NP-Complete Problems

The Clique Problem

The Clique Problem

- A ***clique*** in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E.
  - A ***clique*** is a ***complete*** subgraph of G.
  - The ***size*** of a clique is the number of vertices it contains.
- The ***clique problem*** is the optimization problem of finding a clique of maximum size in a graph.
- The decision problem (**CLIQUE**): Given an undirected graph $G = (V, E)$ and an integer k, is there a clique of size at lease k in G?

Example:



NP-Completeness Proof of CLIQUE (1/4)

1. **Show that CLIQUE is in NP**

   Given $I = \langle G, k \rangle$ and V', verify if V' is a solution to I in polynomial time.

   - Check whether for each pair $u, v \in V'$, the edge (u, v) is in E.

2. **Show that 3-CNF-SAT $\leq_P$ CLIQUE**

   1) Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ be a 3-CNF-SAT formula with k clauses. We will construct a graph G corresponding to ϕ in polynomial time.
      - For each clause $C_i = (l_{i1} \vee l_{i2} \vee l_{i3})$, we create a ***triple*** of three vertices v_{i1}, v_{i2}, v_{i3}.
      - Two vertices v_{ip} and v_{jq} are adjacent iff
        - l_{ip} and l_{jq} are from different clauses, that is, $i \neq j$, and
        - l_{ip} and l_{jq} are ***consistent***, that is, l_{ip} is not the negation of l_{jq}.

NP-Completeness Proof of CLIQUE (2/4)

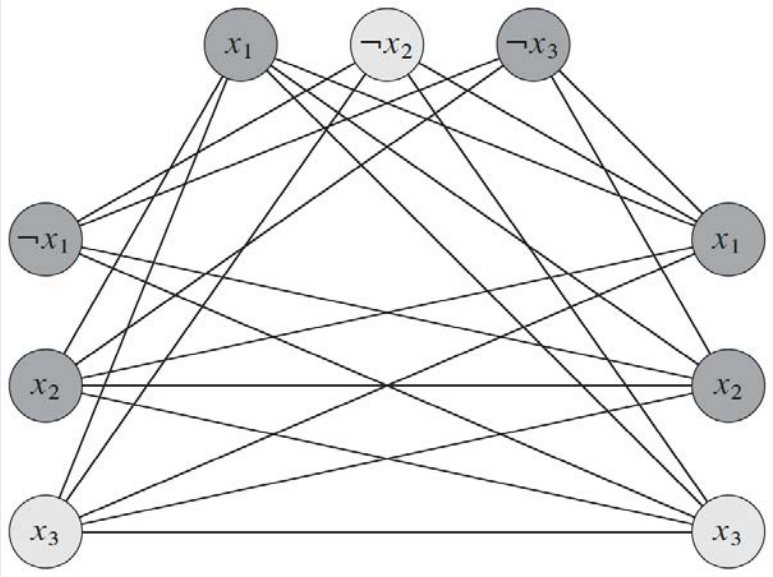
2. Show that 3-CNF-SAT $\leq_P$ CLIQUE (cont'd)

1) Construct a graph G corresponding to ϕ in polynomial time. (cont'd)

Example: $\phi = C_1 \wedge C_2 \wedge C_3$, where

$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee x_2 \vee x_3), C_3 = (x_1 \vee x_2 \vee x_3).$

- The constructed graph $G = (V, E)$ is shown on the right.
- *Recall:* Two vertices v_{ip} and v_{jq} are adjacent iff
  - $i \neq j$ and
  - l_{ip} and l_{jq} are consistent.



NP-Completeness Proof of CLIQUE (3/4)

2. Show that 3-CNF-SAT $\leq_P$ CLIQUE (cont'd)

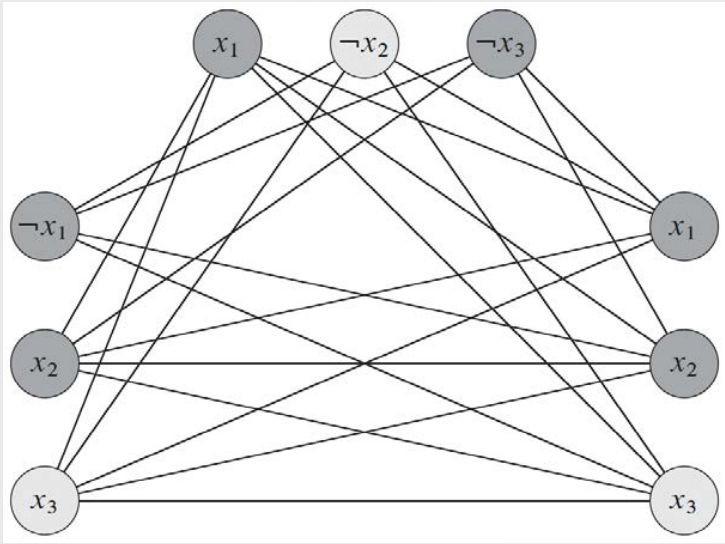
2) Formula ϕ is satisfiable iff G has a clique of size k.

Prove "⇒": Suppose ϕ is satisfiable.

- Each clause C_i has at least one literal being TRUE.
- Pick the vertices corresponding to these literals (one for each clause).
- **Claim:** These vertices form a clique of size k.

Example: $\phi = C_1 \wedge C_2 \wedge C_3$

- $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$
- $C_2 = (\neg x_1 \vee x_2 \vee x_3)$
- $C_3 = (x_1 \vee x_2 \vee x_3)$



NP-Completeness Proof of CLIQUE (4/4)

2. Show that 3-CNF-SAT $\leq_P$ CLIQUE (cont'd)

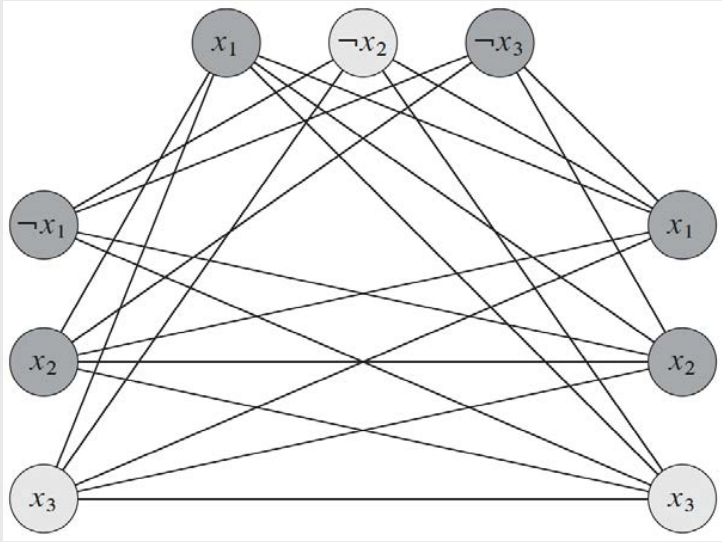
2) Formula ϕ is satisfiable iff G has a clique of size k. (cont'd)

Prove "$\Leftarrow$": Suppose G has a clique V' of size k.

- V' contains exactly one vertex from each triple. (Why?)
- **Claim:** ϕ is a satisfied by assigning TRUE to the literals corresponding to all the vertices in V' and assigning arbitrary truth values to the remaining variables.

Example: $\phi = C_1 \wedge C_2 \wedge C_3$

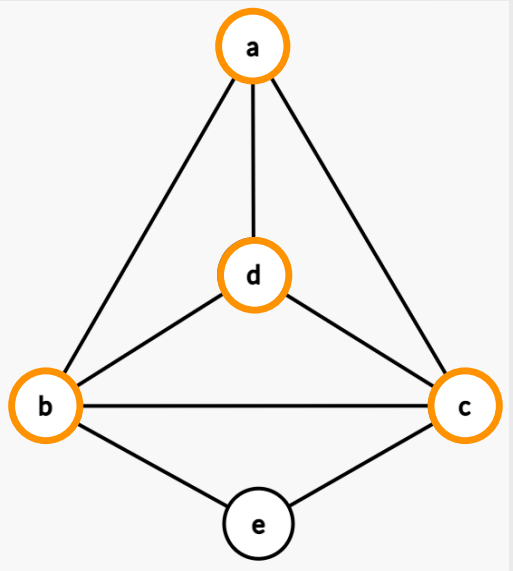
- $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$
- $C_2 = (\neg x_1 \vee x_2 \vee x_3)$
- $C_3 = (x_1 \vee x_2 \vee x_3)$



Independent Set

- An ***independent set*** of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that there is no edge between any pair of vertices in V'.
- The ***independent-set problem*** is the optimization problem of finding an independent set of maximum size in a graph.

Example:



- The decision problem (**IS**): Given a graph $G = (V, E)$ and an integer k, is there an independent set of size at lease k in G?
- **Observation:** If V' is an independent set in G, then it is a clique of the ***complement*** of G, $\bar{G} = (V, \bar{E})$ (in which $e \notin \bar{E}$ iff $e \in E$), and vice-versa.

NP-Completeness Proof of IS

1. **Show that IS is in NP**

   Given $I = \langle G, k \rangle$ and V', verify if V' is a solution to I in polynomial time.

   - Check whether for each pair $u, v \in V'$, $(u, v) \notin E$.

2. **Show that CLIQUE $\leq_P$ IS**

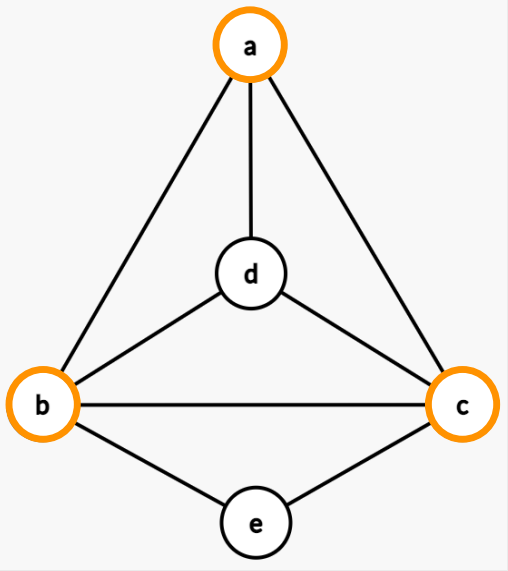
   1) Given an instance $I = \langle G, k \rangle$ of **CLIQUE**, construct the complement of G, $\bar{G} = (V, \bar{E})$. – Can be easily done in polynomial time.
   2) $V' \subseteq V$ is an independent set of G iff V' is a clique of $\bar{G}$.

NP-Complete Problems

The Vertex Cover Problem

Vertex Cover

- A ***vertex cover*** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both).
- The ***vertex-cover problem*** is the optimization problem of finding a vertex cover of minimum size in a given graph.
- The decision problem (**VC**): Given a graph $G = (V, E)$ and an integer k, is there a vertex cover of size at most k in G?
- To prove that **VC** is NP-complete*,
  1. Show that **VC** is in **NP**;
  2. Show that **CLIQUE** $\leq_P$ **VC** or **IS** $\leq_P$ **VC**.



** See Theorem 34.12 on p.1090-1091 of the textbook for the NP-completeness proof with the second part showing that **CLIQUE** $\leq_P$ **VC**.*

NP-Completeness Proof of VC (1/2)

1. **Show that VC is in NP**

   Given $I = \langle G, k \rangle$ and V', verify if V' is a solution to I in polynomial time.

   - Check whether for $(u, v) \in E$, we have $u \in V'$ or $v \in V'$, or both.

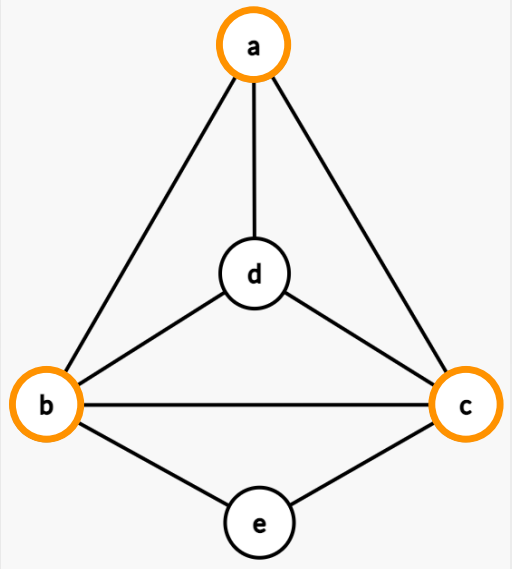
2. **Show that IS $\leq_P$ VC**

   Claim: $V' \subseteq V$ is a vertex cover iff $V - V'$ is an independent set.

   Proof.

   "⇒": V' is a vertex cover ⇒ $V - V'$ is an independent set

   - Suppose $V - V'$ is not an independent set.
   - Then there is a pair $u, v \in V - V'$ s.t. $(u, v) \in E$.
   - ⇒ (u, v) is not covered by V'
   - ⇒ V' is not a vertex cover – A contradiction



NP-Completeness Proof of VC (2/2)

2. Show that IS $\leq_P$ VC (cont'd)

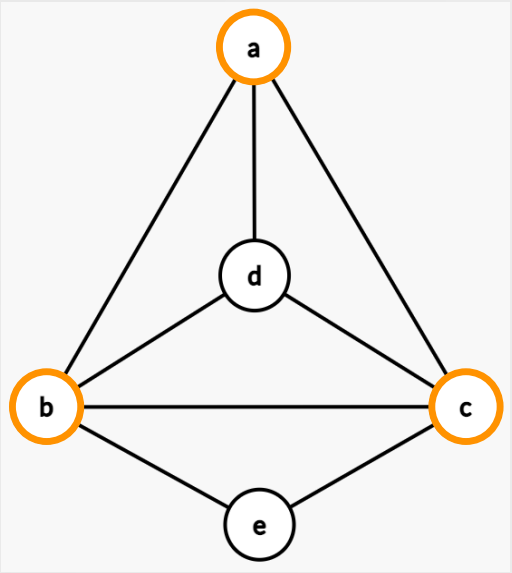
Claim: $V' \subseteq V$ is a vertex cover iff $V - V'$ is an independent set.

Proof (cont'd).

"⇐": $V - V'$ is an independent set ⇒ V' is a vertex cover

- Suppose V' is not a vertex cover.
- Then there is at least one edge $(u, v) \in E$ s.t. $u \notin V'$ and $v \notin V'$.
- ⇒ $u, v \in V - V'$
- ⇒ $V - V'$ is not an independent set – A contradiction

□



The **Claim** implies that: G has a vertex cover of size at most k iff G has an independent set of size at least $n - k$.

NP-Complete Problems

Other NP-Complete Problems

The Hamiltonian-Cycle Problem

- A ***Hamiltonian cycle*** of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V.

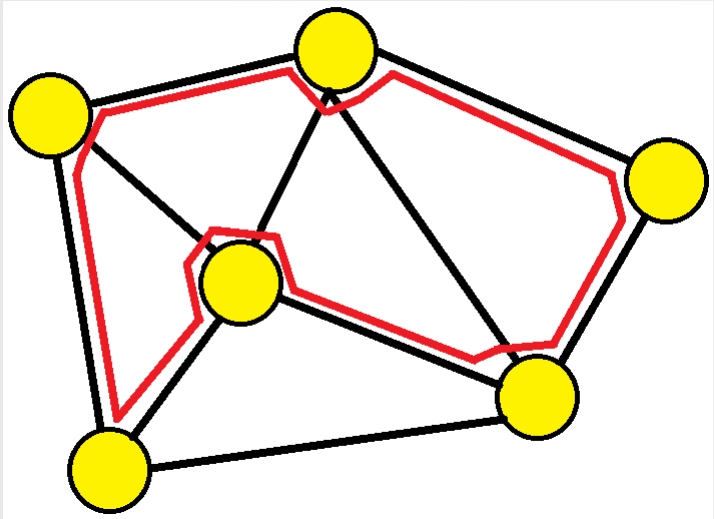
HAM-CYCLE: Given an undirected graph G, is there a ***Hamiltonian cycle*** in G?

To prove that **HAM-CYCLE** is NP-complete*,

1. Show that **HAM-CYCLE** is in **NP**
2. Show that **VC** $\leq_P$ **HAM-CYCLE**

Proofs are omitted.

Example:



** See Theorems 34.13 on p.1091 of the textbook for the NP-completeness proofs.*

The Traveling-Salesman Problem

The traveling-salesman problem (optimization problem):

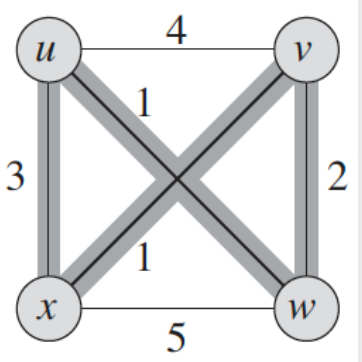
- **Input:** A complete graph $G = (V, E)$, with n vertices (representing n cities), a nonnegative integer cost $c(i, j)$ on each edge $(i, j) \in E$ (for a salesman to travel from city i to city j)
- **Output:** A minimum-cost tour (Hamiltonian cycle).

The decision problem (**TSP**): Given a complete graph G, a nonnegative integer cost on each edge, and a nonnegative integer k, is there a tour (Hamiltonian cycle) with cost at most k?

To prove that **TSP** is NP-complete*,

1. Show that **TSP** is in **NP**
2. Show that **HAM-CYCLE** $\leq_P$ **TSP**

Example:



** See Theorems 34.14 on p.1096 of the textbook for the NP-completeness proofs.*

HAM-CYCLE $\leq_P$ TSP

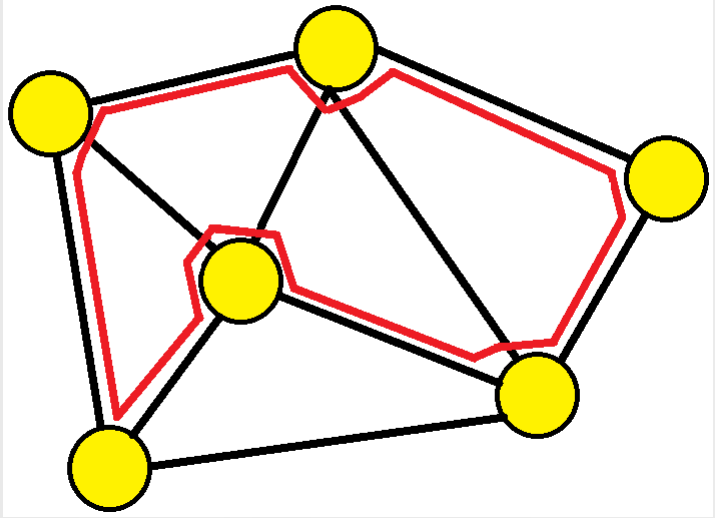
An instance I_{HC} of **HAM-CYCLE**: An undirected graph $G = (V, E)$.

Transform I_{HC} to an instance I_{TSP} of **TSP**:

- $G' = (V, E')$, where E' contains every edge (i, j) for $i, j \in V$
- $c(i,j) = \begin{cases} 0, & \text{if } (i,j) \in E \\ 1, & \text{if } (i,j) \notin E \end{cases}$ and $k = 0$

Claim: G has a Hamiltonian cycle iff G' has a tour of cost at most 0.

Example:



The Subset-Sum Problem

SUBSET-SUM: Given a finite set S of positive integers and an integer target $t > 0$, is there a subset $S' \subseteq S$ whose elements sum to t?

- **Example 1:** $S = \{1, 2, 3, 4\}$ and $t = 8$.

  Answer: Yes. $S' = \{1, 3, 4\}$

- **Example 2:** $S = \{1, 2, 3, 4\}$ and $t = 11$.

  Answer: No.

To prove that **SUBSET-SUM** is NP-complete*,

1. Show that **SUBSET-SUM** is in **NP**
2. Show that **3-CNF-SAT** $\leq_P$ **SUBSET-SUM**

Proofs are omitted.

** See Theorems 34.15 on p.1097 of the textbook for the NP-completeness proof.*

The Integral Knapsack Problem

Integral Knapsack (Optimization problem):

- **Input:** n items, with integer weight w_i and integer value v_i for each item, a knapsack with capacity W, and an integer V.
- **Output:** A most valuable subset of items with total weight $\leq W$.

The decision problem (**INT-KNAPSACK**): Is there a subset of items with total weight at most W and total value at least V?

To prove that **INT-KNAPSACK** is NP-complete,

1. Show that **INT-KNAPSACK** is in **NP**
2. Show that **SUBSET-SUM** $\leq_P$ **INT-KNAPSACK**

SUBSET-SUM $\leq_P$ INT-KNAPSACK

An instance I_{SS} of **SUBSET-SUM**:

- A set of positive integers $S = \{s_1, s_2, \cdots, s_n\}$ and an integer $t > 0$

Transform I_{SS} to an instance I_{IK} of **INT-KNAPSACK**:

- n items, with integer weight $w_i = s_i$ and integer value $v_i = s_i$ for each item, a knapsack with capacity $W = t$, and an integer $V = t$.

Claim: There is a subset $S' \subseteq S$ whose elements sum to t iff there is a subset of items with total weight at most t and total value at least t.

Example: $S = \{1, 2, 3, 4\}$, $t = 8$.

- $\sum_{i=1}^{n} w_i \leq W \Leftrightarrow \sum_{i=1}^{n} s_i \leq t$
- $\sum_{i=1}^{n} v_i \geq V \Leftrightarrow \sum_{i=1}^{n} s_i \geq t$

$\Big\} \sum_{i=1}^{n} s_i = t$

Ways to Deal with NP-completeness

- A problem is **NP-complete** means: (currently) unlikely to find a polynomial time algorithm to solve it exactly for all instances.
- To tackle an NP-complete problem, we can
  1. Solve it exactly, but in exponential time
     - for small instances, may be perfectly satisfactory
     - for important special cases, may be satisfactory
  2. Recognize special structure for the important special cases
     - design polynomial time algorithms if possible
  3. Design polynomial time algorithms to solve it approximately
     - no guaranteed performance - heuristics
     - provably guaranteed performance - ***approximation algorithms***
- We will talk about ***approximation algorithms*** for optimization problems whose decision versions are NP-complete.

Thank you!
Questions?