2025 Fall CSE5025

Combinatorial Optimization

组合优化

Instructor: 刘晟材

Lecture 2-2: NPC, NP-Hardness, and NP-Hard Optimization Problems

A Quick Recap on P, NP, NPC

The **class P** consists of all **decision problems** that are solvable in polynomial time. That is, there exists an algorithm that will **decide** in polynomial time if any given instance is a yes-instance or a no-instance.

The **class NP** consists of all **decision problems** such that, for **each yes-instance,** there exists a **certificate** that can be **verified** in **polynomial time**.

The **class NPC** of **NP-Complete problems** consists of all **decision problems** L such that:

1) $L \in$ NP,

2) For every $L' \in$ NP, $L' \leq_P L$.

A Quick Recap on Polynomial-Time Reductions

Definition: A **Polynomial-Time Reduction** from L_1 to L_2 is a transformation f with the following properties:

- f transforms an instance x of L_1 into an instance $f(x)$ of L_2 s.t.

  $f(x)$ is a yes-instance for L_2 if and only if x is a yes-instance for L_1.
- $f(x)$ is computable in polynomial time in size(x).

If such an f exists, we say that $\boldsymbol{L_1}$ **is polynomial-time reducible (多项式时间内可规约) to** $\boldsymbol{L_2}$, and write:

$$\boldsymbol{L_1 \leq_P L_2}.$$

Implications: If $L_1 \leq_P L_2$, then L_1 is no harder than L_2.

Agenda for Today's Lecture

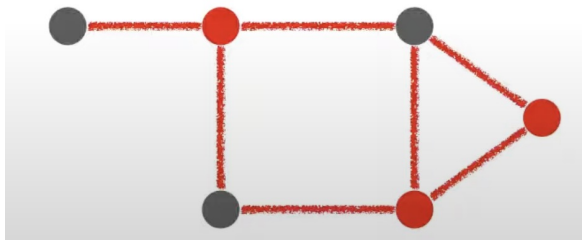
Continuing from our last lecture, today we will focus on

- Decision Vertex Cover (DVC) $\leq_P$ Decision Independent Set (DIS)
- Ladner Theorem
- Proving a Problem is NP-Complete
- NP-Hardness
- Decision Problems vs. Optimization Problems
- Proving an Optimization Problem is NP-Hard

DVC and DIS

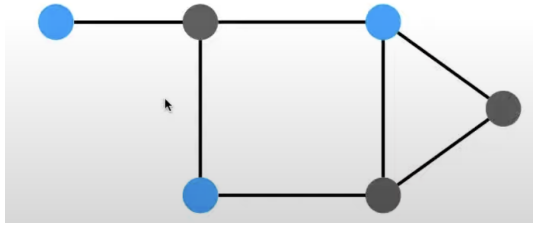
Given a graph $G = (V, E)$ with n vertices.

Vertex Cover



A vertex cover (VC, 定点覆盖集) is a subset of vertices $S \subseteq V$, such that every edge in E is connected by at least one vertex in S.
Decision VC (DVC): Does G have a VC of size **at most** k?

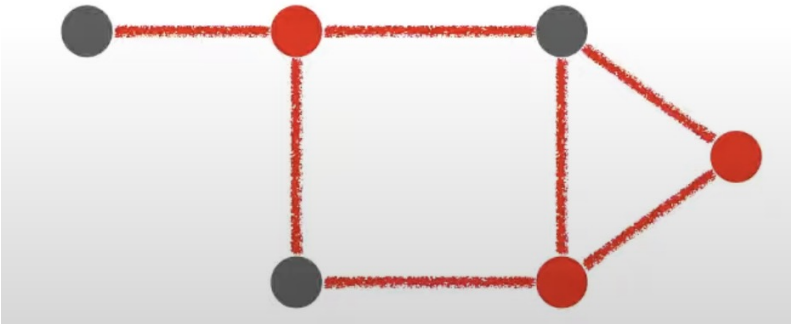
Independent Set



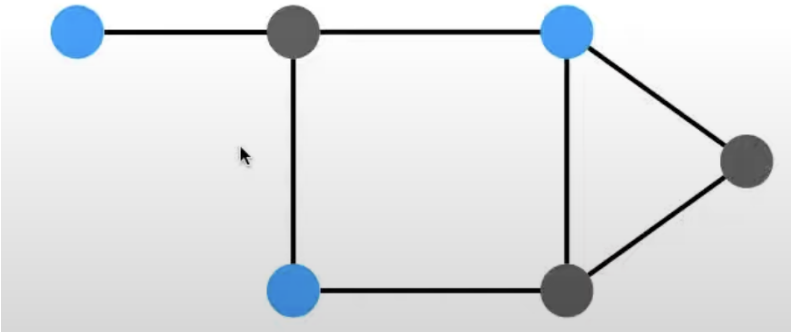
An independent set (IS, 独立集) is a subset of vertices $S \subseteq V$, such that no two vertices in S are connected by an edge.
Decision IS (DIS): Does G have an IS of size **at least** k?

DVC $\leq_P$ DIS (1)

Vertex Cover



Independent Set



Core Claim: a subset of vertices $S \subseteq V$ of the graph G is IS if and only if its complement $V \backslash S$ is a VC of G.

This means: G has a VC of size k if and only if G has an IS of size $|V| - k$.

DVC $\leq_P$ DIS (2)

The Polynomial-Time Reduction Algorithm

Input: An instance of DVC: a graph G and an integer k.

Transformation: Create an instance of DIS with graph $G' = G$ and integer $k' = n - k$:

Proof:

1) This transformation takes constant time $O(1)$.

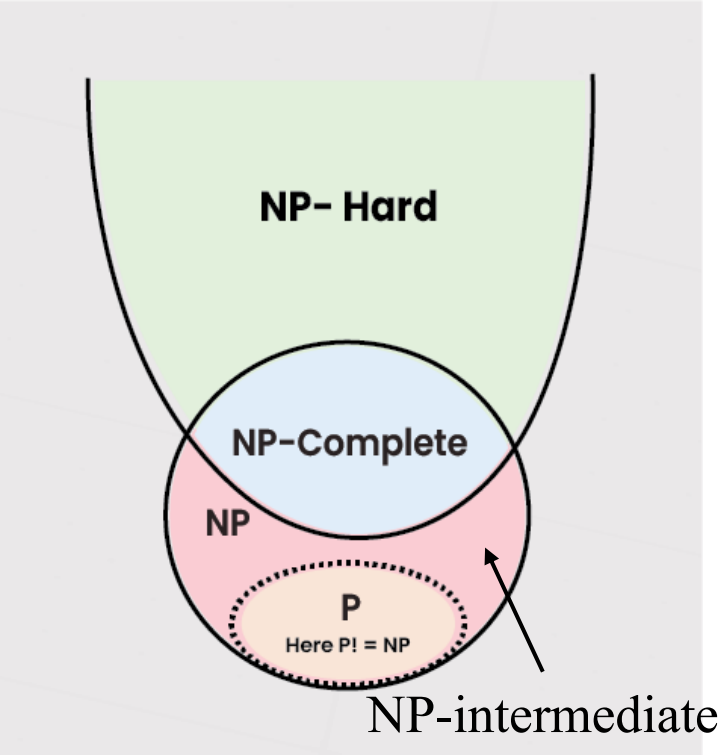
2) An yes-instance of DVC means G has a VC, i.e., denoted by S, and $|S| \leq k$. As proved earlier, $V \backslash S$ is an IS of G' (since $G' = G$), and $|V \backslash S| \geq n - k$. Hence, G' has an IS of size at least $n - k$, which means the created DIS instance is an yes-instance.

3) A no-instance of DVC means every VC of G, denoted by S, satfying $|S| \geq k + 1$. As proved earlier, $V \backslash S$ is an IS of G' (since $G' = G$), and $|V \backslash S| \leq n - k - 1$. Hence, every IS of G' is of size at most $n - k - 1$, which means the created DIS instance is a no-instance.

Ladner's Theorem

Problems that are in NP but are neither in the class P nor NP-Complete are called NP-intermediate, and the class of such problems is called **NPI**

Ladner's theorem, shown in 1975 by Richard E. Ladner, is a result asserting that, if P ≠ NP, then NPI is not empty; that is, NP contains problems that are neither in P nor NP-complete.



Proving a Problem L is NPC

Two steps:

1) Show $L \in$ NP.

2) Show that $L' \leq_P L$ for a suitable $L' \in$ NPC

We will study some specific NP-Complete problems

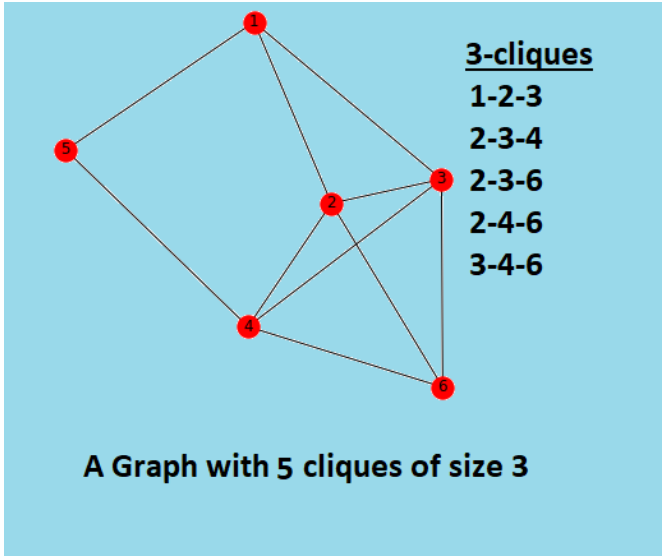
1) **DCLIQUE**: by showing **3-SAT** $\leq_P$ **DCLIQUE**

2) **DVC**: by showing **DCLIQUE** $\leq_P$ **DVC**

3) **DIS**: by showing **DVC** $\leq_P$ **DIS**

Problem: CLIQUE

Clique (团): given an undirected graph $G = (V, E)$, a clique is a subset of vertices of V, i.e., $S \subseteq V$, such that **every** two distinct vertices in S are **adjacent**. In other words, a clique is a complete subgraph of G (a vertex is a clique of size 1, an edge is a clique of size 2).



Clique Problem (最大团问题): Find a clique of maximum size in a graph.

Decision Problem: DCLIQUE

The Decision Clique Problem (DCLIQUE, 团判定问题): given an undirected graph $G = (V, E)$ and an integer k, does G have a clique with k vertices?

Theorem: DCLIQUE ∈ NPC

Proof: We need to show two things:

1) That DCLIQUE ∈ NP and

2) That there is some $L \in$ NPC such that $L \leq_P$ DCLIQUE.

Proving DCLIQUE ∈ NPC (1)

Proving 1) is easy.

A certificate will be a set of vertices $S \subseteq V, |S| = k$ that is a possible clique. To check that S is a clique, check that all edges (u, v) with $u \neq v, u, v \in S$ are in E. This can be done in time $O(|V|^2)$ if the edges are kept in an adjacency matrix.

To prove 2) we will show that **3-SAT** $\leq_P$ **DCLIQUE**. This is hard.

We will do this by building a "gadget" that allows a reduction from the 3-SAT problem (on logical formulas) to the DCLIQUE problem (on graphs).

Proving DCLIQUE ∈ NPC (2)

For a fixed k, consider Boolean formulas in k-conjunctive normal form (k-CNF, k-合取范式): $C = C_1 \wedge C_2 \wedge \cdots \wedge C_n$; each C_i (called clause, 子句) is of the form: $C_i = C_{i,1} \vee C_{i,2} \vee \cdots \vee C_{i,k}$, where each $C_{i,1}$ (called literal, 字面值) is a Boolean variable or the negation of a variable.

An example of a 3-CNF formula is:

$$C = (X_1 \vee \neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee X_2 \vee X_3)$$

We will define a polynomial-time transformation f from 3-SAT (3-CNF SAT) to DCLIQUE:

$$f: C \rightarrow (G, k)$$

Proving DCLIQUE ∈ NPC (3)

Suppose that C is a 3-SAT formula with n clauses, i.e., $C = C_1 \wedge C_2 \wedge \cdots \wedge C_n$.

We start by setting k=n and construct graph $G = (V, E)$:

1) For each clause $C_i = c_{i,1} \vee c_{i,2} \vee c_{i,3}$, create 3 vertices, v_1^i, v_2^i, v_3^i, in V so G has $3n$ vertices. We **label these vertices with the corresponding variable or variable negation**. (Note that many vertices might share the same label)

2) Create an edge between vertices v_j^i and $v_{j'}^{i'}$ if and only if the following conditions hold:

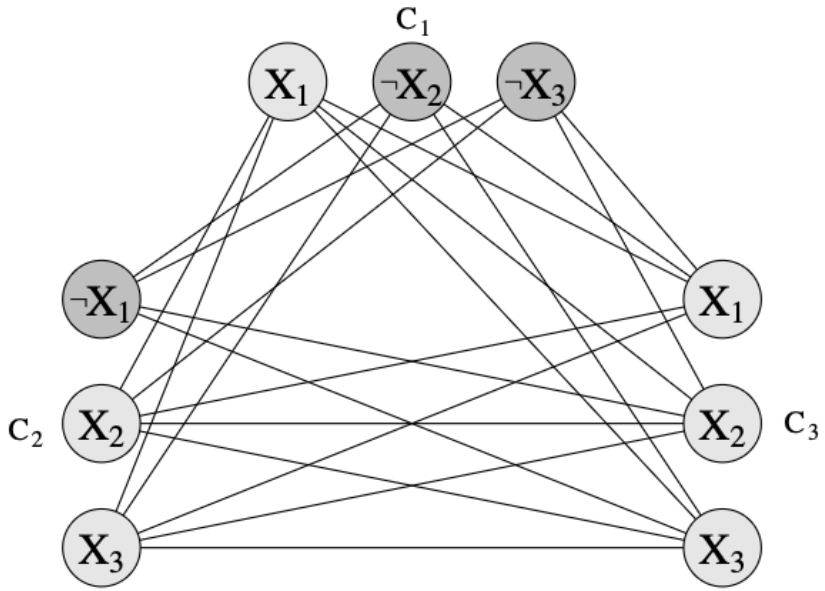
(a) v_j^i and $v_{j'}^{i'}$ are in different triples (clauses), i.e., $i \neq i'$, and

(b) v_j^i is not the negation of $v_{j'}^{i'}$

Proving DCLIQUE ∈ NPC (4)

Here is a formula $C = C_1 \wedge C_2 \wedge C_3$ and its corresponding graph:

$$C_1 = (X_1 \vee \neg X_2 \vee \neg X_3), C_2 = (\neg X_1 \vee X_2 \vee X_3), C_3 = (X_1 \vee X_2 \vee X_3)$$



Note that the assignment $X_3 = \text{True}, X_2 = \text{False}$ satisfies C (independent of the value of X_1). This corresponds to the clique of size 3 comprising the $\neg X_2$ node in C_1, the X_3 node in C_2 and the X_3 node in C_3. So C is satisfiable and G has a 3-clique.

Proving DCLIQUE ∈ NPC (5)

Lemma: A 3-CNF formula C with k clauses is satisfiable if and only if $f(C) = (G, k)$ is a Yes-instance to DCLIQUE. (proof provided later)

Note that the graph G has $3k$ vertices and at most $3k(3k-1)/2$ edges and can be built in $O(k^2)$ time: f is a Polynomial-time reduction.

We have therefore just proven that

$$\text{3-SAT} \leq_P \text{DCLIQUE}$$

Since we already know that 3−CNF ∈ NPC and have seen that DCLIQUE ∈ NP we have just proven that **DCLIQUE ∈ NPC**.

Proof of the Lemma (1)

Part 1: C **is Satisfiable** $\Longrightarrow$ G **has a** k**-Clique**

Assumption: The formula C is satisfiable.

1) This means there is a truth assignment that makes at least one literal in each of the k clauses TRUE.
2) From each clause, select exactly one literal that is TRUE under this assignment. This gives us a set of k TRUE literals.
3) Consider the k vertices in the graph G that correspond to these k selected literals. Let's call this set of vertices V'.

Question: Is V' a k-clique?

Proof of the Lemma (2)

Let's check if any two vertices in V' have an edge between them.

- Are they from different clauses? Yes. By our selection method, we picked exactly one vertex from each of the k clauses.
- Are their literals non-contradictory? Yes. All selected literals are TRUE under a single, consistent assignment. A variable and its negation cannot both be true at the same time.

Conclusion: Since any pair of vertices in V' satisfies both conditions for an edge, they are all connected to each other. Therefore, V' is a clique of size k.

Proof of the Lemma (3)

Part 2: G has a k-Clique $\Longrightarrow$ C is Satisfiable

Assumption: The graph G contains a clique of size k. Let's call the set of vertices in this clique V'.

1) **Observation 1**: Vertices from the same clause are not connected. This means the k vertices in the clique must have come from k different clauses—exactly one vertex per clause.
2) **Observation 2**: By definition of an edge, no two vertices in the clique can represent contradictory literals (like X_1 and $\neg X_1$).

We can use this clique to define a valid truth assignment.

Proof of the Lemma (4)

For each vertex in the clique V', take its corresponding literal and set the associated variable to make that literal TRUE.

- Example: If a vertex for $\neg X_2$ is in the clique, we set variable X_2 to FALSE.

Why this works:

- The assignment is consistent. Observation 2 guarantees we never try to set a variable to both TRUE and FALSE.
- The assignment satisfies the formula. Observation 1 guarantees that every clause has exactly one literal represented in our clique. By our rule, we have made this literal TRUE, thus satisfying every clause.

Conclusion: The existence of a k-clique in G allows us to construct a satisfying assignment for C.

Proving DVC ∈ NPC (1)

Definition: Given a graph $G = (V, E)$ with n vertices, a vertex cover (VC) is a subset of vertices $S \subseteq V$, such that every edge in E is connected by at least one vertex in S.

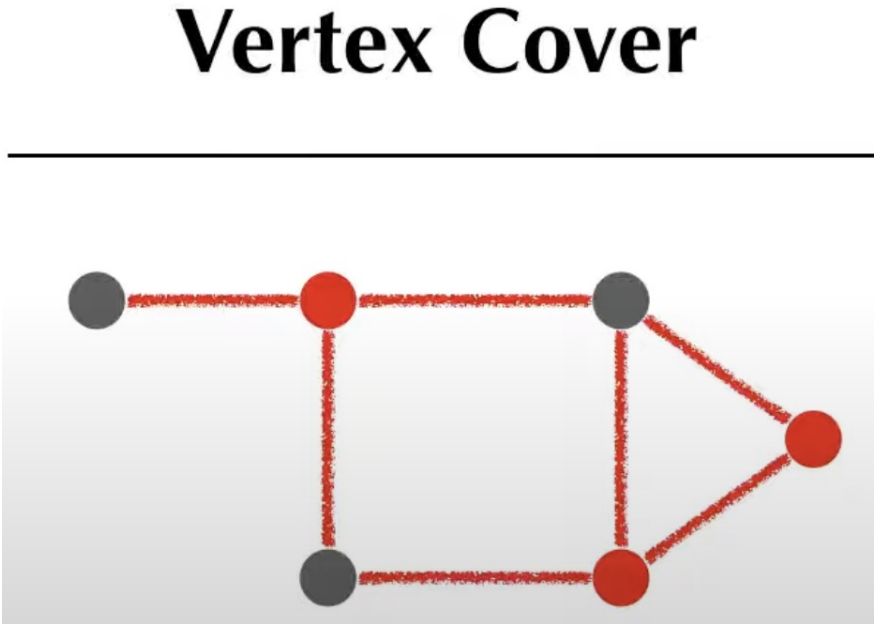
Decision VC (DVC): Given $G = (V, E)$ and an integer k, does G have a VC of size at most k?

Theorem: **DVC ∈ NPC**

Proof: We will show two things:

1) That DVC ∈ NP and

2) That DCLIQUE $\leq_P$ DVC.



Proving DVC ∈ NPC (2)

Claim: DVC ∈ NP.

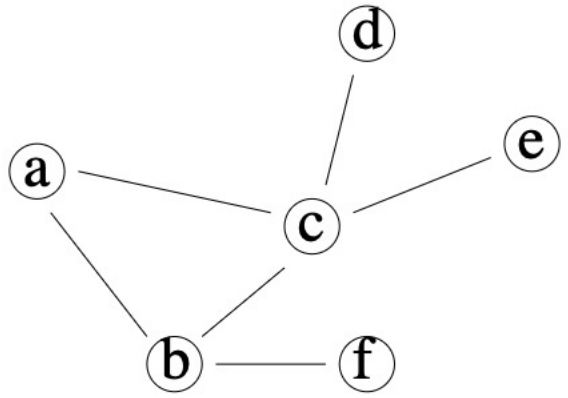
Proof: A certificate will be a set S of $\leq k$ vertices. The brute force method to check whether S is a vertex cover takes time $O(k|E|)$. As $k|E| < n|E|$, the time to verify is $O(n|E|)$. So a certificate can be verified in polynomial time.

Proving DVC ∈ NPC (3)

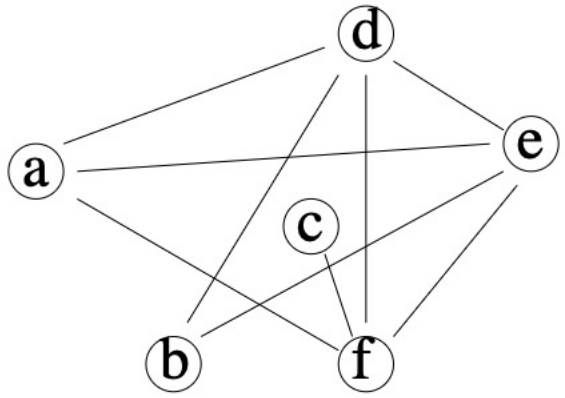
Claim: DCLIQUE $\leq_P$ DVC.

The **complement** of a graph $G = (V, E)$ is defined by $\bar{G} = (V, \bar{E})$, where

$$\bar{E} = \{(u, v) | u, v \in V, u \neq v, (u, v) \notin E\}$$



Graph G



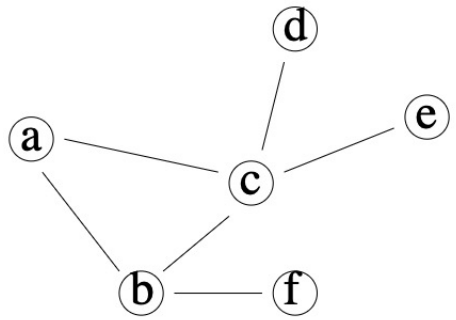
Complement of G

Proving DVC ∈ NPC (4)

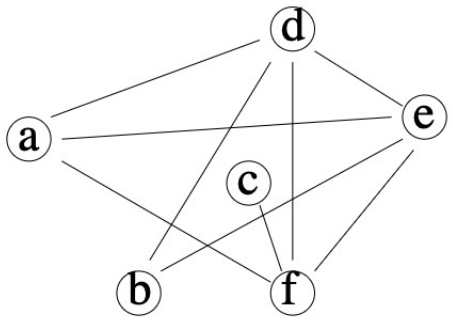
Lemma: A graph G has a vertex cover of size k if and only if the complement graph $\bar{G}$ has a clique of size $|V| - k$.

Proof: Let S be a vertex cover in G and let $S' = V \backslash \mathrm{S}$. If u and v are distinct vertices in S', then they are not connected by an edge in E and so $(u, v) \in \bar{E}$. Hence S' are the vertices of a clique in $\bar{G}$.

Similarly, if S' is a clique in $\bar{G}$, then $S = V \backslash S'$ is a vertex cover in G.



Graph G



Complement of G

Proving DVC ∈ NPC (5)

Proof of DCLIQUE $\leq_P$ DVC:

Let $\bar{k} = |V| - k$. We define a transformation f from DCLIQUE to DVC:

$$f: (G, k) \rightarrow (\bar{G}, \bar{k})$$

- f can be computed in $O(|V|^2)$ time
- (G, k) is a Yes-instance for DCLIQUE if and only if $(\bar{G}, \bar{k})$ is an Yes-instance for DVC (by the Lemma in the previous slide)

Hence f is a polynomial-time reduction from DCLIQUE to DVC.

Remark: In this very special case f is also invertible and polynomial-time reduction from DVC to DCLIQUE as well.

Proving DIS ∈ NPC (1)

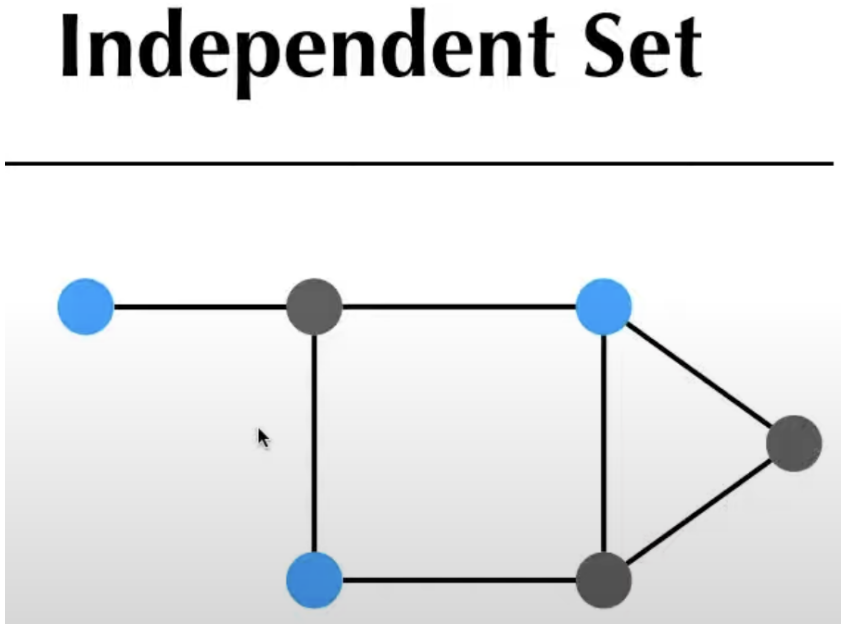
Definition: Given $G = (V, E)$, an Independent Set (IS) is a subset of vertices $S \subseteq V$, such that no two vertices in S are connected by an edge.

Decision IS (DIS): Given $G = (V, E)$ and an integer k, does G have an IS of size at least k?

Theorem: **DIS ∈ NPC**

Proof: We have shown DVC $\leq_P$ DIS, and only need to show that

1) DIS ∈ NP.



Proving DIS ∈ NPC (2)

Claim: DIS ∈ NP.

Proof: A certificate will be a set S of $\geq k$ vertices. The brute force method to check whether S is a independent set takes time $O(k^2)$. As $k^2 < |V|^2$, the time to verify is $O(|V|^2)$. So a certificate can be verified in polynomial time.

What about Optimization Problems?

The theory of NP-Completeness revolves around **decision problems,** because it's easier to compare the difficulty of decision problems than that of optimization problems.

At first glance, this might seem unhelpful since what we really care about are optimization problems. We're interested in finding an **optimal solution** to our problem (the optimization version), not whether such a **solution exists** (decision version).

Decision versus Optimization Problems (1)

However, **solving a decision problem in polynomial time will often permit us to solve the corresponding optimization problem in polynomial time** (using a polynomial number of calls to the decision problem).

So, discussing the difficulty of decision problems is **often** equivalent to discussing the difficulty of optimization problems.

Decision versus Optimization Problems (2)

Here are the two problems and third related one

1) **VC**: Given an undirected graph G, find the **minimal-size vertex cover**.

2) **DVC**: Given an undirected graph G and k, is there a vertex cover of size at most k?

3) **MVC**: Given an undirected graph G, find the **size** of the minimal vertex cover.

Supposing that algorithm $DVC(G,k)$ solves DVC in polynomial time, consider the following algorithm for solving MVC:

Algorithm MVC(G)

1. $k = 0$;
2. **while** (**not** $DVC(G,k)$) $k = k + 1$;
3. **return**(k);

It calls $DVC(G,k)$ at most $|V|$ times so, if there is a polynomial time algorithm for DVC, then the above algorithm for MVC is also polynomial.

Decision versus Optimization Problems (3)

Here is an algorithm for calculating $VC(G)$ that uses the algorithm for MVC.

First set $k = MVC(G)$ and then run $VC(G, k)$:

Algorithm $VC(G, t)$ to find VC of size t

1. Check all vertices in G to find the first vertex u such that $MVC(G_u) == t - 1$;

(G_u: remove vertex u and its associated edges from G)

2. Output u;

3. **if** $(t > 1)$ $VC(G_u, t-1)$;

Note that this algorithm calls MVC at most $|V|^2$ times so, if MVC is polynomial in $size(G)$, then so is VC.

We already saw that if DVC is polynomial in $size(G)$, so is MVC, so we've just shown that if we can solve DVC in polynomial time, we can solve VC in polynomial time.

NP-Hardness (1)

Definition: A problem L is called **NP-hard** if every problem in NP can be **reduced** to L in polynomial time.

Equivalent Definition: A problem L is **NP-hard** if some problem in NPC can be **reduced** to L in polynomial time.

Note L does not need to be in NP. It's possible that a solution of NP-Hard problem cannot be verified in polynomial time.

Note L does not need to be a decision problem.

NP-Hardness (2)

Intuitive Definition: The Oracle Analogy

A more intuitive way to understand this is through the concept of an "oracle". Imagine you are given an oracle that can solve problem *L* in **a single step**.

A problem *L* is **NP-hard** if, with the help of this oracle, you can write an algorithm that solves **any problem in NP (or some problem in NPC)** in polynomial time.

The term "hard" in NP-hard means **"at least as hard as the hardest problems in NP" (NPC).**

NP-Hardness (3)

In general, the optimization versions of NP-Complete problems are NP-Hard.

For example, recall

1) VC: Given an undirected graph G, find the minimal-size vertex cover.

2) DVC: Given an undirected graph G and k, is there a vertex cover of size at most k?

Proof: If we can solve the optimization problem VC with an oracle solver $VC(G)$, we can easily solve the decision problem DVC. Simply run $VC(G)$ for **only once** and find a minimal vertex cover S. Now, given (G, k), by checking whether $k \geq |S|$ (in constant time). If $k \geq |S|$ answer Yes, if not, answer No.

Two Types of Reductions - A Conceptual Distinction

For NP-Hardness, we **use a more general reduction (Cook Reduction, also called Turing reduction)** than the one used for NP-Completeness.

Karp Reduction (for NP-Completeness)

- Transforms an instance of one decision problem to an instance of another.
- x is a yes-instance for Problem A if and only if $f(x)$ is a yes-instance for Problem B.
- Scope: Strictly Decision Problem → Decision Problem.

Cook Reduction (for NP-Hardness)

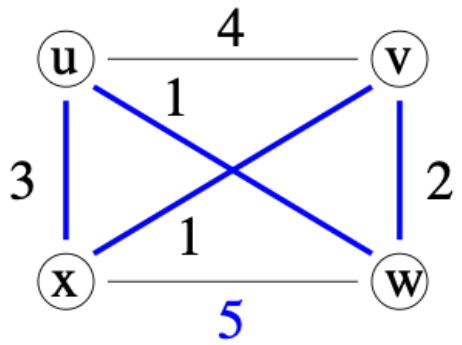
- Uses a solver (oracle) for Problem B as a subroutine to build a new algorithm for Problem A.
- Scope: Universal. Can be applied from Any Problem → Any Problem (e.g., Decision → Optimization).

Reductions Compared: Karp vs. Cook

	Karp Reduction	Cook Reduction
Core Mechanism	**Instance Transformation:** Designs a function f that maps an instance x of problem A to an instance $f(x)$ of problem B.	**Algorithm Construction:** Designs an algorithm for problem A that uses a solver for problem B as a subroutine (oracle).
Scope	**Strictly limited:** Only from a **Decision Problem** to another **Decision Problem**.	**Universal:** Can be from **Any Problem** to **Any Problem** (Decision, Optimization, Count, etc.).
Primary Use Case	Proving a problem is **NP-Complete**.	Proving a problem is **NP-Hard**.

TSP and DTSP

Optimization Problem TSP: Given a complete weighted undirected graph with n vertices, **find a Hamiltonian cycle of least weight**.



Tour (Hamiltonian cycle)
with minimum cost 7

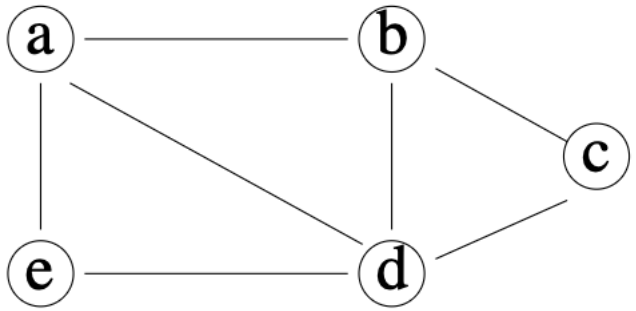
Decision Problem: Decision TSP (DTSP)

Given a complete weighted undirected graph with n vertices, and a bound B, is there a Hamiltonian cycle of weight $\leq B$?

DHamCyc

Hamiltonian Cycle: Given an undirected graph $G = (V, E)$, a cycle of graph G is called Hamiltonian if it contains every vertex exactly once.

Example:



Find a Hamiltonian cycle for this graph

Decision problem DHamCyc: Does G have a Hamiltonian cycle?

Proving TSP is NP-Hard

In the rest of this lecture, we will show the **TSP is NP-Hard**.

What we will do:

1) Prove DTSP is NP-Complete by assuming DHamCyc is NP-Complete and showing DHamCyc $\leq_P$ DTSP (Carp Reduction).

2) Prove DTSP can be reduced to TSP in polynomial time (Cook Reduction).

Proving TSP is NP-Hard (1)

Claim: DHamCyc ∈ NPC.

Proof:

1) In the last lecture we have proved DHamCyc ∈ NP

2) Using a complicated "gadget" we can construct a polynomial-time reduction proving that DVC $\leq_P$ DHamCyc (omitted here)

Proving TSP is NP-Hard (2)

Claim: DTSP ∈ NP.

Proof:

Certificate Verification in $O(n)$: an ordering of the n vertices in G (corresponding to their order along the Hamiltonian Cycle), i.e., $v_1, v_2, \ldots, v_n$.

- check that the cycle contains each vertex once;
- Check that the total weight $\leq B$

Proving TSP is NP-Hard (3)

Claim: DHamCyc $\leq_P$ DTSP

An instance of DHamCyc: an undirected graph $G = (V, E)$

An instance of DTSP: an undirected complete graph G', a weight function c, and a bound B.

Proof:

1) We define the transformation f: DHamCyc $\rightarrow$ DTSP by $f(G) = (G', c, B)$:

$$G' = (V, \{(u, v) | u, v \in V, u \neq v\}),$$

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \neq E \end{cases}$$

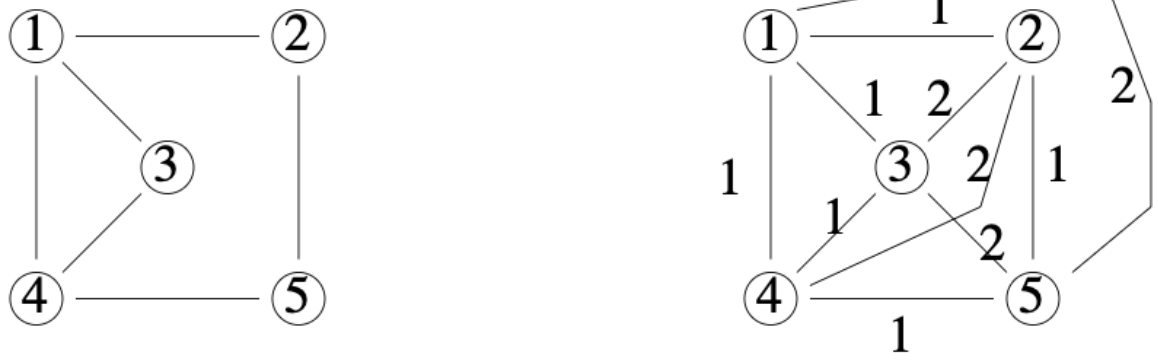
$$B = |V|$$

Proving TSP is NP-Hard (4)

2) f can be computed in $O(|V|^2)$ and so is polynomial

Mapping f:

DHamCyc instance → DTSP instance



Graph G = (V, E)

Graph G' = (V, E')
weights c (on edges)
bound B = 5

Proving TSP is NP-Hard (5)

3) If G has a Hamiltonian cycle, then G' has a Hamiltonian cycle of weight at most n, i.e., $|V|$.

Suppose $s = \langle v_1, v_2, \ldots, v_n \rangle$ is a Hamiltonian cycle of G.

Then $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, n-1$ and $(v_n, v_1) \in E$.

Hence $c(v_i, v_{i+1}) = 1$ for $i = 1, 2, \ldots, n-1$ and $c(v_n, v_1) = 1$.

Therefore, s is a Hamiltonian cycle of weight n in G′.

Proving TSP is NP-Hard (6)

4) If G' has a Hamiltonian cycle of weight at most n, then G has a Hamiltonian cycle.

Suppose $s = \langle v_1, v_2, \ldots, v_n \rangle$ is a Hamiltonian cycle of G' of weight at most n. Since the cycle has n edges and each edge has weight at least 1, the total weight must be exactly n and each edge must have weight 1.

Therefore, all the edges, including (v_i, v_{i+1}) for $i = 1, 2, \ldots, n-1$ and (v_n, v_1), belong to E

Therefore, s is a Hamiltonian cycle in G.

Combining with 3), it is proved that DHamCyc $\leq_P$ DTSP

Proving TSP is NP-Hard (7)

Claim: DTSP can be reduced to TSP in polynomial time

An instance of TSP: an undirected complete graph G, a weight function c

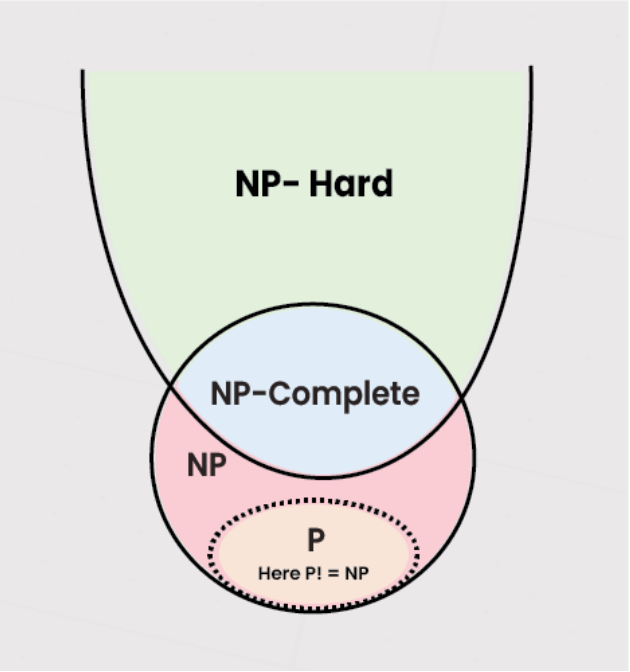
An instance of DTSP: an undirected complete graph G, a weight function c, and a bound B.

Proof: If we can solve TSP with an oracle solver $TSP(G, c)$, we can easily solve any given DTSP instance (G, c, B). Given (G, c, B), simply run $TSP(G, c)$ for **only once** and find a minimal-weight Hamiltonian cycle S. Calculate the weight of S (in $O(|V|)$) and check whether it is not larger than B (in $O(1)$). If yes answer Yes, if not, answer No.

Conclusion

We have introduced the following key concepts:

- Input size of problems
- Decision problems (判定问题)
- Polynomial time algorithms
- The Class P, NP, NPC, reductions between decision problems
- **Proving Problems are NPC**
- The Class NP-Hard
- **Proving Problems are NP-Hard**



All the optimization problems studied in this course are NP-Hard