L11: NP Completeness

Yanlin Zhang & Wei Wang
HKUST(GZ)

May 4, 2025

TOC

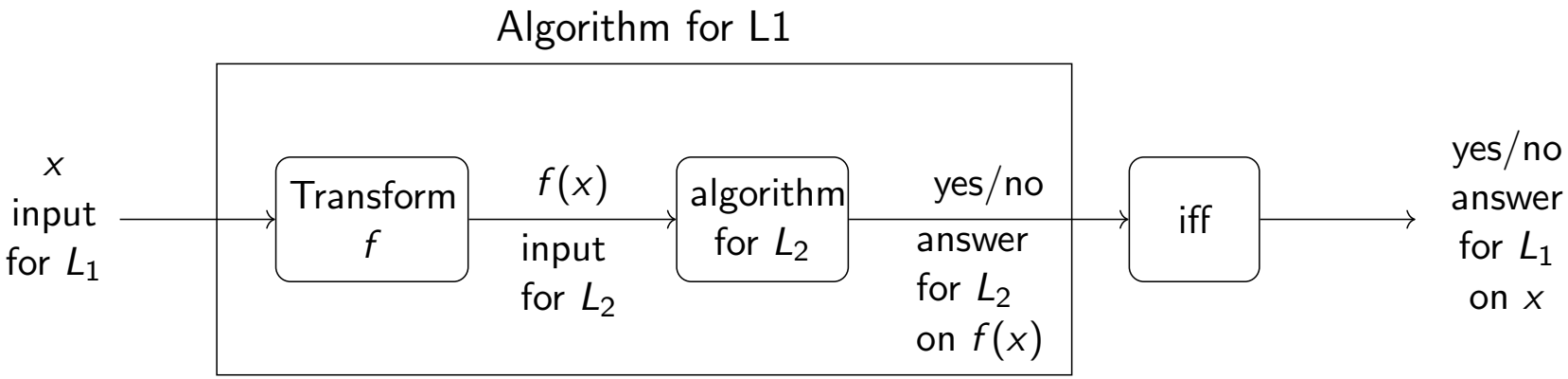
- Polynomial-time reductions.
- The class NPC.
- Proving that problems are NPC. SAT, VERTEX COVER
- Optimization vs. Decision problems

What is Reduction?

Let L_1 and L_2 be two decision problems.

Suppose algorithm A_2 solves L_2. That is, if y is an input for L_2 then algorithm A_2 will answer Yes or No depending upon whether $y \in L_2$ or not.

The idea is to find a transformation f from L_1 to L_2 so that the algorithm A_2 can be part of an algorithm A_1 to solve L_1.



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 input x for L_1 into an input $f(x)$ for L_2 s.t.

  $f(x)$ is a **yes-input for** L_2
  if and only if
  x is a **yes-input for** L_1.

- $f(x)$ is computable in polynomial time (in $size(x)$).

If such an f exists, we say that L_1 is **polynomial-time reducible to** L_2, and write $L_1 \leq_P L_2$.

Polynomial-Time Reductions

Question

What can we do with a polynomial time reduction $f : L_1 \rightarrow L_2$?

Answer

Given an algorithm A_2 for the decision problem L_2, we can develop an algorithm A_1 to solve L_1.

In particular (proof on next slide):

- if A_2 is a **polynomial time algorithm for** L_2 and $L_1 \leq_P L_2$ then we can construct **a polynomial time algorithm for** L_1.

Polynomial-Time Reduction $f : L_1 \to L_2$

Theorem

If $L_1 \leq_P L_2$ and $L_2 \in \mathrm{P}$, then $L_1 \in \mathrm{P}$.

Proof.

$L_2 \in \mathrm{P}$ means that we have a polynomial-time algorithm A_2 for L_2. Since $L_1 \leq_P L_2$, we have a polynomial-time transformation f mapping input x for L_1 to an input for L_2. Combining these, we get the following polynomial-time algorithm for solving L_1:

1. take input x for L_1 and compute $f(x)$;
2. run A_2 on input $f(x)$, and return the answer found (for L_2 on $f(x)$) as the answer for L_1 on x.

Each of Steps (1) and (2) takes polynomial time. So the combined algorithm takes polynomial time. Hence $L_1 \in \mathrm{P}$. □

Warning

We have just seen

Theorem

If $L_1 \leq_P L_2$ and $L_2 \in \mathrm{P}$, then $L_1 \in \mathrm{P}$.

Note that this does not imply that
If $L_1 \leq_P L_2$ and $L_1 \in \mathbf{P}$, then $L_2 \in \mathbf{P}$.
This statement is not true.

Reduction between Decision Problems

Lemma (Transitivity of the relation $\leq_P$)

If $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then $L_1 \leq_P L_3$.

Proof.

Since $L_1 \leq_P L_2$, there is a polynomial-time reduction f_1 from L_1 to L_2.
Similarly, since $L_2 \leq_P L_3$ there is a polynomial-time reduction f_2 from L_2 to L_3.

Note that $f_1(x)$ can be calculated in **time** polynomial in $size(x)$.
In particular this implies that $size(f_1(x))$ is polynomial in $size(x)$.
$f(x) = f_2(f_1(x))$ can therefore be calculated in time polynomial in $size(x)$.
Furthermore x is a **yes-input for** L_1 if and only if $f(x)$ **is a yes-input for** L_3 (why). Thus the combined transformation defined by $f(x) = f_2(f_1(x))$ is a polynomial-time reduction from L_1 to L_3. Hence $L_1 \leq_P L_3$. □

Example of Polynomial-Time Reduction

Definition: CYC

Does an undirected graph G have a cycle?

Definition: TRIPLE

Does a triple (n, e, t) of nonnegative integers satisfy $e \neq n - t$?

Definition: Reduction f

To show CYC $\leq_p$ TRIPLE, we define f to be

$$f : G \to (n, e, t),$$

where n, e and t are the number of vertices, edges and connected components of G respectively.

Example Continued

We will show that

1. G has a cycle if and only if $f(G) \in$ TRIPLE.
2. $f(G)$ can be calculated in time polynomial in $size(G) = n + e$.

This proves that

$$\textbf{CYC} \leq_P \textbf{TRIPLE}.$$

Example Continued

Theorem

G has a cycle if and only if $f(G) \in$ TRIPLE

follows from

Lemma

A graph G has no cycles if and only if $e = n - t$

Proof.

$\Rightarrow$ [i.e, If G has no cycles, then $e = n - t$]

- If G has no cycles, then each connected component G_i has no cycles.
- Since G_i has no cycles and is connected, G_i is a tree.
  - Then for each tree G_i, we know: $e_i = n_i - 1$.

$$e = \sum_i e_i = \sum_i (n_i - 1) = \left(\sum_i n_i\right) - \left(\sum_i 1\right) = n - t$$

Example Continued

Proof.

$\Leftarrow$ [i.e., If $e = n - t$, then G has no cycles]
Proof by contradiction. (Hint: Assume G has at least one cycle. Create G' from G by removing one edge from the cycle, and use the $\Rightarrow$ part) $\square$

To show that $f(G)$ can be calculated in time polynomial in $size(G)$ we need to solve

Problem

Design a polynomial time algorithm that, given undirected graph $G = (V, E)$, computes the number of connected components of G.

Possible solutions: Modify BFS or DFS, adding a counter. The next page gaves an $O(|E| + |V|)$ time algorithm.

Example Continued

One solution: A modified DFS that runs in time $O(n+e)$:

```
1  COMP(G){
2       for each u in V
3            color[u]=W;
4       number=0;
5       for each u in V
6            if (color[u]==W){
7                 number=number+1;
8                 DFSVisit(u);
9            }
10      return(number);
11 }
12 DFSVisit(u){
13      color[u]=G;
14      for each v in adj[u]
15           if (color[v]==W)
16                DFSVisit(v);
17      color[u]=B;
18 }
```

Node color change: white $\rightarrow$ gray $\rightarrow$ black

Example Continued

Using the algorithm on the previous slide we see that we can calculate t in time polynomial in $size(G)$. Since we can also calculate e and n in polynomial time, **we can calculate $f(G)$ in polynomial time**.

Recall that we saw G **has a cycle if and only if $f(G) \in$ TRIPLE so G is a yes-input for CYC if and only if $f(G)$ is a yes-input for TRIPLE**.

Combining these two facts shows that f is a Polynomial-Time Reduction from CYC to TRIPLE or

$$\textbf{CYC} \leq_P \textbf{TRIPLE}$$

Finally: The Class NP-Complete (NPC)

We have finally reached our goal of introducing

Definition

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

- $L \in \mathrm{P}$;
- for every $L' \in \mathrm{NP}$, $L' \leq_P L$.

Intuitively, NPC consists of all the **hardest** problems in NP.

Note: it is not obvious that there exist any such problems at all. There are an infinite number of problems in NP. How can we prove that some problem is at least as hard as all of them?

We will see later that there are actually many such problems.

NP-Completeness and Its Properties

The major reason we are interested in NP-Completeness is the following theorem which states that

- either **all** NP-Complete problems are polynomial time solvable
- or **all** NP-Complete problems are not polynomial time solvable.

Theorem

Suppose that L is NPC. If there is a polynomial-time algorithm for L, then there is a polynomial-time algorithm for every $L' \in$ NP.

Proof: By the theorem on Page 6.

Theorem

Suppose that L is NPC. If there is no polynomial-time algorithm for L, then there is no polynomial-time algorithm for any $L' \in NPC$.

Proof: By the previous conclusion.

The Classes P, NP, co-NP, and NPC

Proposition

$P \subseteq NP$.

Simple proof omitted.

Question

Is $NPC \subseteq NP$?

Yes, by definition!

Question

Is $P = NP$?

Open problem! Probably very hard. It is generally believed that $P \neq NP$. Proving this (or the opposite) would win you the US$1,000,000 Clay Prize.

Question

Is NP = co-NP?

Open problem! Probably also very hard

Note: if NP $\neq$ co-NP, then P $\neq$ NP (why?)

Proving that problem L is NPC-Complete

Steps:

1. Show $L \in$ NP.
2. Show that $L' \leq_P L$ for a suitable $L' \in$ NPC.

Question

How do we get one problem in NPC to start with?

Answer: We need to prove, **from scratch** that one problem is in NPC.

Question

Which problem is "suitable"?

Answer: There is no general procedure to determine this. You have to be knowledgeable, clever and (sometimes) lucky.
c.f., Garey and Johnson's book *Computers and Intractability: A Guide to the Theory of NP-Completeness*.

Proving that problems are NPC-Complete

In the rest of this lecture we will discuss some specific NP-Complete problems.

1. **SAT** and **3-CNF-SAT**.
   We will assume that they are NP-Complete. (From textbook)

2. **Decision Vertex Cover DVC**:
   by showing DCLIQUE $\leq_P$ DVC
   The reduction used is very natural.

The Satisfiability Problem is NPC

Cook (1971): SAT $\in$ NPC.

Remark: Since Cook showed that SAT $\in$ NPC, thousands of problems have been shown to be in NPC using the reduction approach described earlier.

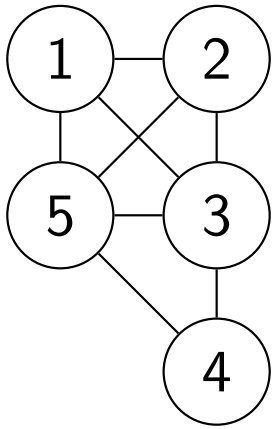
Remark: With a little more work we can also show that 3-CNF-SAT $\in$ NPC as well.

Note: For this course, you only need to know the validity of the two facts above but do not need to know how to prove they are correct.

Problem: CLIQUE

Definition: Clique

A **clique** in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each pair $u, v \in V'$ is connected by an edge $(u, v) \in E$. In other words, a clique is a **complete** subgraph of G.



Find a clique with 4 vertices

Problem: **CLIQUE**

Find a clique of maximum size in a graph.

NPC Problem: DCLIQUE

Definition: DCLIQUE

The Decision Clique Problem (**DCLIQUE**): Given an undirected graph G and an integer k, determine whether G has a clique with k vertices.

Theorem

DCLIQUE $\in$ NPC.

Proof: We need to show two things.
(a) That **DCLIQUE** $\in$ **NP** and
(b) That there is some $L \in$ **NPC** such that

$$L \leq_P \textbf{DCLIQUE}.$$

Proof that DCLIQUE $\in$ NPC

Theorem

DCLIQUE $\in$ NPC.

Proof: We need to show two things.
(a) That **DCLIQUE** $\in$ **NP** and
(b) That there is some $L \in$ **NPC** such that

$$L \leq_P \textbf{DCLIQUE}.$$

Proving (a) is easy. A certificate will be a set of vertices $V' \subseteq V$, $|V'| = k$ that is a possible clique. To check that V' is a clique all that is needed is to check that all edges (u, v) with $u \neq v$, $u, v \in V'$, are in E. This can be done in time $O(|V|^2)$ if the edges are kept in an adjacency matrix (and even if they are kept in an adjacency list – how?).

Proof that DCLIQUE $\in$ NPC

Theorem

DCLIQUE $\in$ NPC.

Proof: We need to show two things.
(a) That **DCLIQUE** $\in$ **NP** and
(b) That there is some $L \in$ **NPC** such that

$$L \leq_P \textbf{DCLIQUE}.$$

To prove (b) we will show that

$$\textbf{3-CNF-SAT} \leq_P \textbf{DCLIQUE}.$$

This will be the hard part.
We will do this by building a 'gadget' that allows a reduction from the 3-CNF-SAT problem (on logical formulas) to the DCLIQUE problem (on graphs).

Proof that DCLIQUE $\in$ NPC (cont)

Recall that the input to 3-CNF-SAT is a logical formula φ of the form

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

where each C_i is of the form

$$C_i = y_{i,1} \vee y_{i,2} \vee y_{i,3}$$

where each $y_{i,j}$ is a variable or the negation of a variable.
As an example

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

We will define a polynomial transformation f from 3-CNF-SAT to DCLIQUE

$$f : \varphi \to (G, k)$$

that builds a graph G and integer k such that
(G, k) **is a Yes-input to DCLIQUE if and only if** φ **is a Yes-input to 3-CNF-SAT**, i.e, φ is satisfiable.

Proof that DCLIQUE $\in$ NPC (cont)

Suppose that φ is a 3-CNF-SAT formula with n clauses, i.e., $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$. We start by setting $k = n$.
We now construct graph $G = (V, E)$.

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

(II) We create an **edge** between vertices v_j^i and $v_{j'}^{i'}$ if and only if the following two conditions hold:

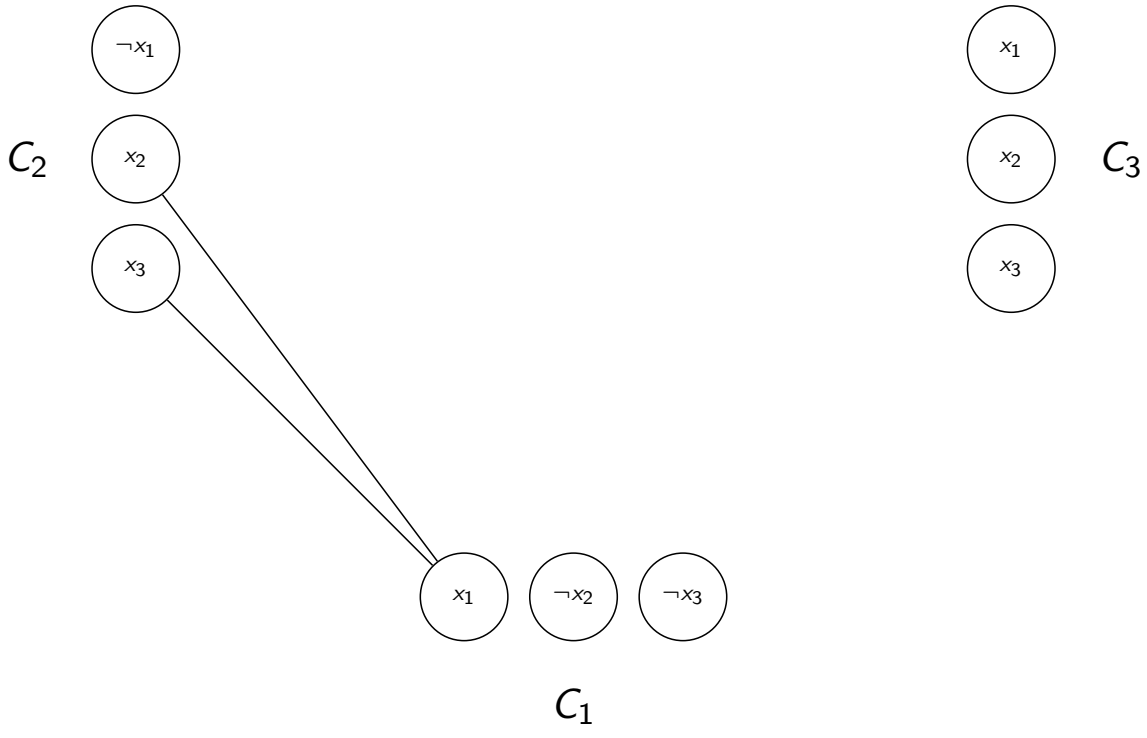
a. v_j^i and $v_{j'}^{i'}$ are in different triples, i.e., $i \neq i'$, and
b. v_j^i is **not** the **negation** of $v_{j'}^{i'}$.

An example of this construction is given in the next slide.

Proof that DCLIQUE $\in$ NPC (cont)

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

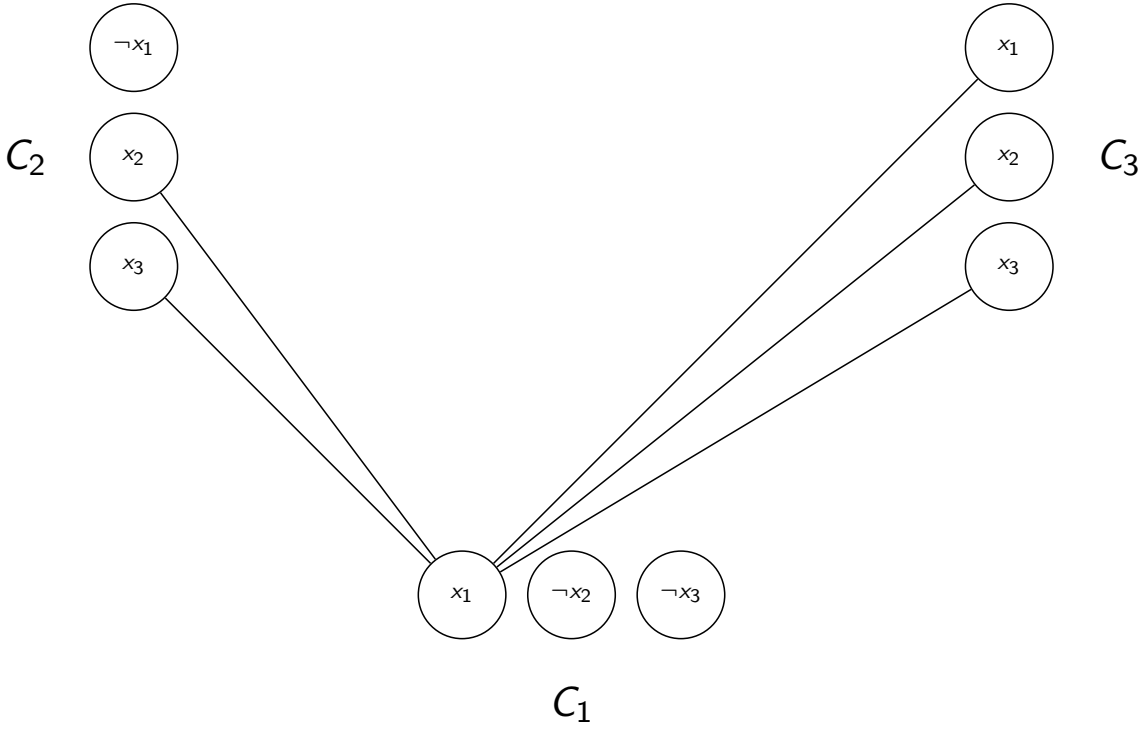
$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee x_2 \vee x_3), C_3 = (x_1 \vee x_2 \vee x_3)$



Proof that DCLIQUE $\in$ NPC (cont)

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

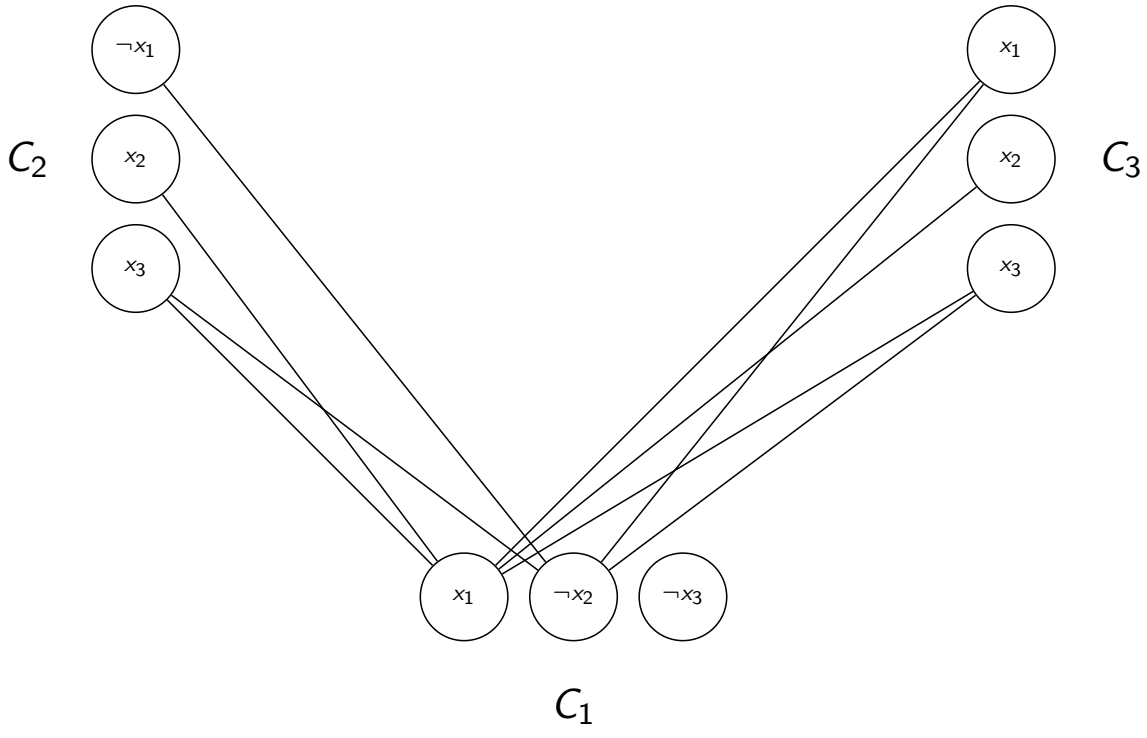
$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee x_2 \vee x_3), C_3 = (x_1 \vee x_2 \vee x_3)$



Proof that DCLIQUE $\in$ NPC (cont)

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

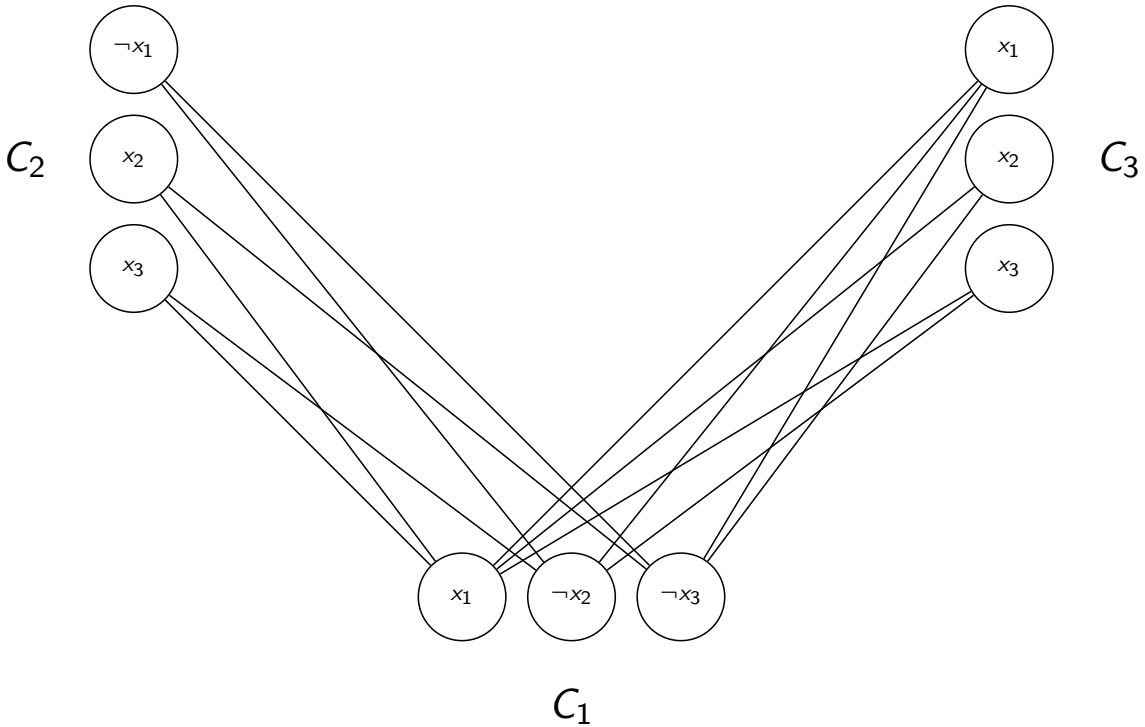
$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee x_2 \vee x_3), C_3 = (x_1 \vee x_2 \vee x_3)$



Proof that DCLIQUE $\in$ NPC (cont)

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

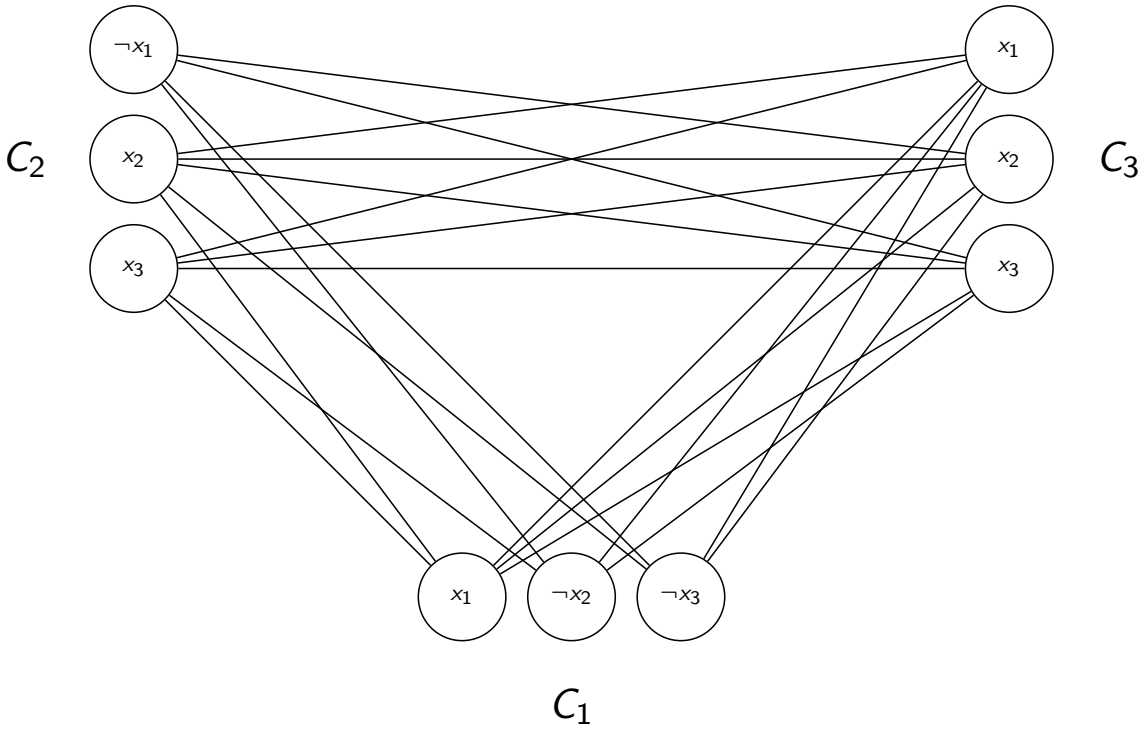
$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee x_2 \vee x_3), C_3 = (x_1 \vee x_2 \vee x_3)$



Proof that DCLIQUE $\in$ NPC (cont)

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

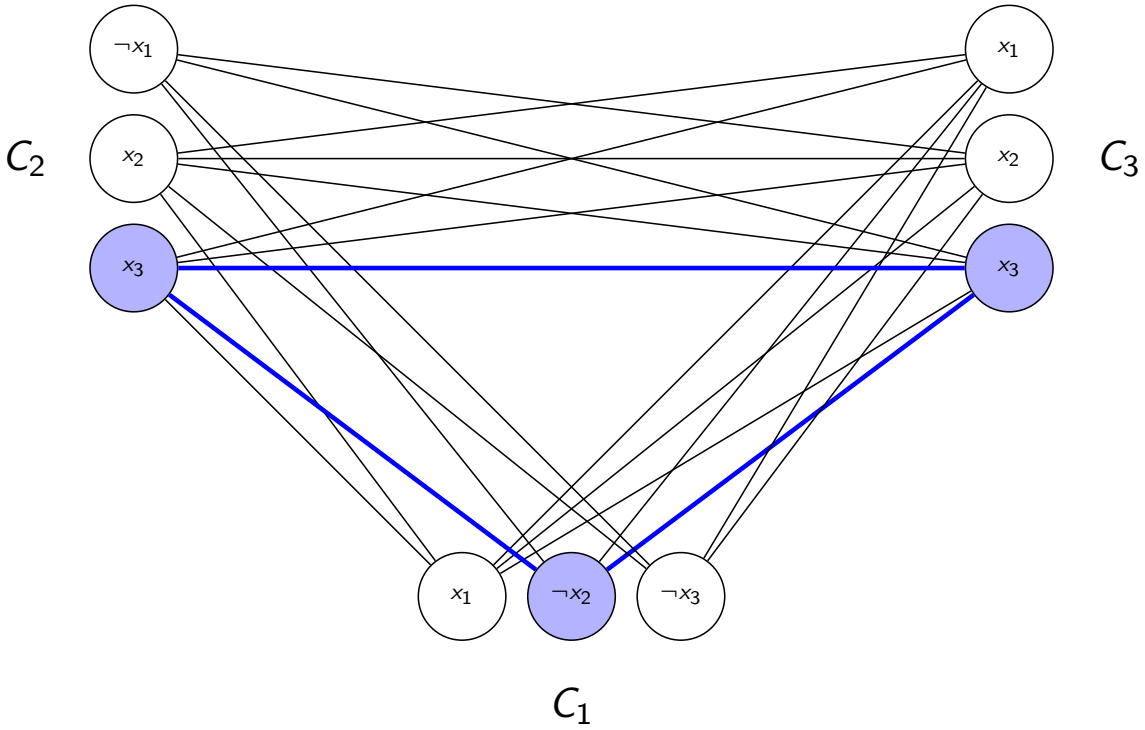
$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee x_2 \vee x_3), C_3 = (x_1 \vee x_2 \vee x_3)$



Proof that DCLIQUE $\in$ NPC (cont)

Here is a formula $\varphi = C_1 \wedge C_2 \wedge C_3$ 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 $x_3 = T, x_2 = F$ satisfies φ (independent of the value of x_1). This corresponds to the clique of size 3 as marked blue above.

So φ is satisfiable and G has a 3-clique.

Proof that DCLIQUE $\in$ NPC (cont)

Theorem

A 3-CNF-SAT formula φ with k clauses is satisfiable **if and only if** $f(\varphi) = (G, k)$ is a Yes-input to DCLIQUE.

Proof.

($\Rightarrow$): choose one literal in each clause to make it true; then obvious
($\Leftarrow$): obvious $\square$

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

We have therefore just proven that

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

Since we already know that 3-CNF-SAT $\in$ NPC and have seen that **DCLIQUE** $\in$ **NP** we have just proven that **DCLIQUE** $\in$ **NPC**.

Problem: VC

Definition: Vertex Cover

A **vertex cover** of G is a set of vertices such that every edge in G is incident to at least one of these vertices.

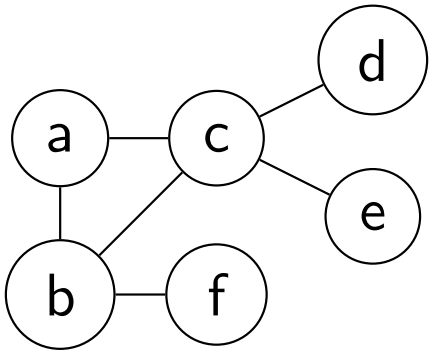


Figure: Find a vertex cover of G of size two

Problem: Vertex Cover Problem (VC)

Given a graph G, find a vertex cover of G of minimum size.

NPC Problem: DVC

Problem: The Decision Vertex Cover Problem (DVC)

Given a graph G and integer k, determine whether G has a vertex cover with k vertices.

Theorem

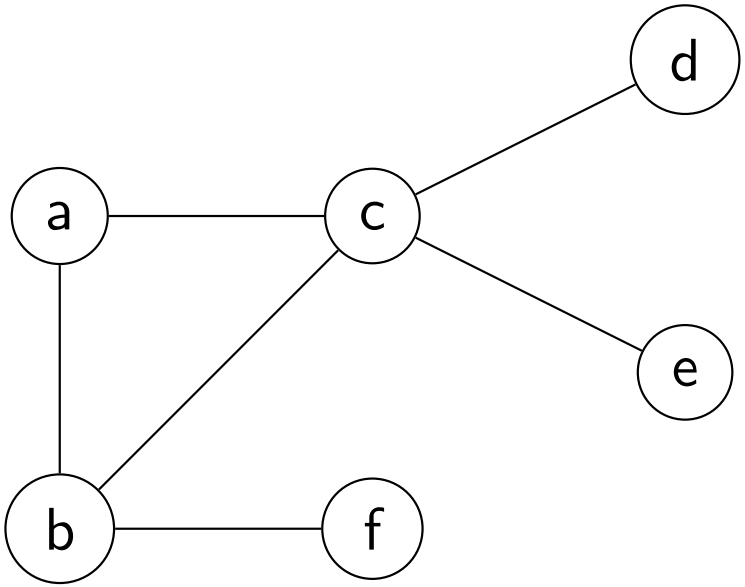
DVC $\in$ NPC.

Proof: In an earlier Lecture, we showed that DVC $\in$ NP.
We show that DCLIQUE $\leq_P$ DVC. The conclusion then follows from the fact that DCLIQUE $\in$ NPC. A proof of DCLIQUE $\leq_P$ DVC will be given in the next three slides.

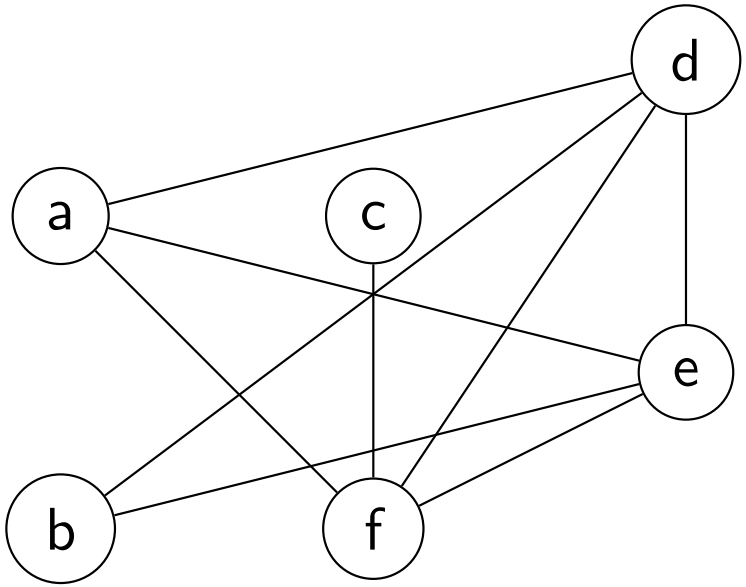
DVC $\in$ NPC: Complement of a Graph

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

$$\overline{E} \stackrel{\text{def}}{=} \{ (u, v) \mid u, v \in V, u \neq v, (u, v) \notin E \}$$



(a) Graph G



(b) Complement of G

DVC $\in$ NPC: Vertex Cover Lemma

Lemma

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

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

Similarly, if V'' is a clique in $\overline{G}$, then $V' = V \setminus V''$ is a vertex cover in G.

Proof: DVC $\in$ NPC

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

$$f : (G, k) \mapsto (\overline{G}, \overline{k})$$

- f can be computed (that is, $\overline{G}$ and $\overline{k}$ can be determined) in time $O(|V|^2)$ time.

- f is a reduction, i.e., (G, k) is a Yes-input for DCLIQUE if and only if $f(G, k)$ is a Yes-input 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.

Decision versus Optimization Problems

The theory of NP-Completeness revolves around decision problems. It was set up this way because it's easier to compare the difficulty of decision problems than that of optimization problems.

At first glance, this might seem unhelpful since we usually don't care at all about decision 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 (cont)

However, being able to solve 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 really equivalent to discussing the difficulty of optimization problems.

In the next two slides we see an example of this phenomenon for VERTEX COVER by showing that having a polynomial algorithm for Decision Vertex Cover (DVC) would yield a polynomial algorithm for finding a minimal Vertex Cover.

Decision versus Optimization Problems (cont)

Here are the two problems and third related one

Definition: VC

Given undirected graph G find a minimal size vertex cover.

Definition: DVC

Given undirected graph G and k, is there a vertex cover of size k?

Definition: MVC

Given an undirected graph G, find the size of a minimal vertex cover.

Algorithm to Solve MVC

Suppose that $DVC(G, k)$ returns Yes if G has a vertex cover of size k and No, otherwise.
Consider the following algorithm for solving MVC:

Algorithm to Solve MVC

Suppose that $DVC(G, k)$ returns Yes if G has a vertex cover of size k and No, otherwise.
Consider the following algorithm for solving MVC:

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

Note that MVC calls DVC at most $|V|$ times so, if there is a polynomial time algorithm for DVC, then our algorithm for MVC is also polynomial.

Algorithm to Solve VC

Given the MVC algorithm, here is an algorithm for calculating $VC(G)$

First set $k = MVC(G)$ and then run $VC(G, k)$ which is defined by

```
1  VC(G, t){    // find VC of size t
2      check all vertices in G to find the first
3        vertex u such that MVC(G_{-u}) = t-1;
4      // such a vertex exists, why?
5      Output u;
6      if (t > 1)
7          VC(G_{-u}, t-1);
8      }
```

Q: What is G_{-u}?

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-Hard Problems

Definition

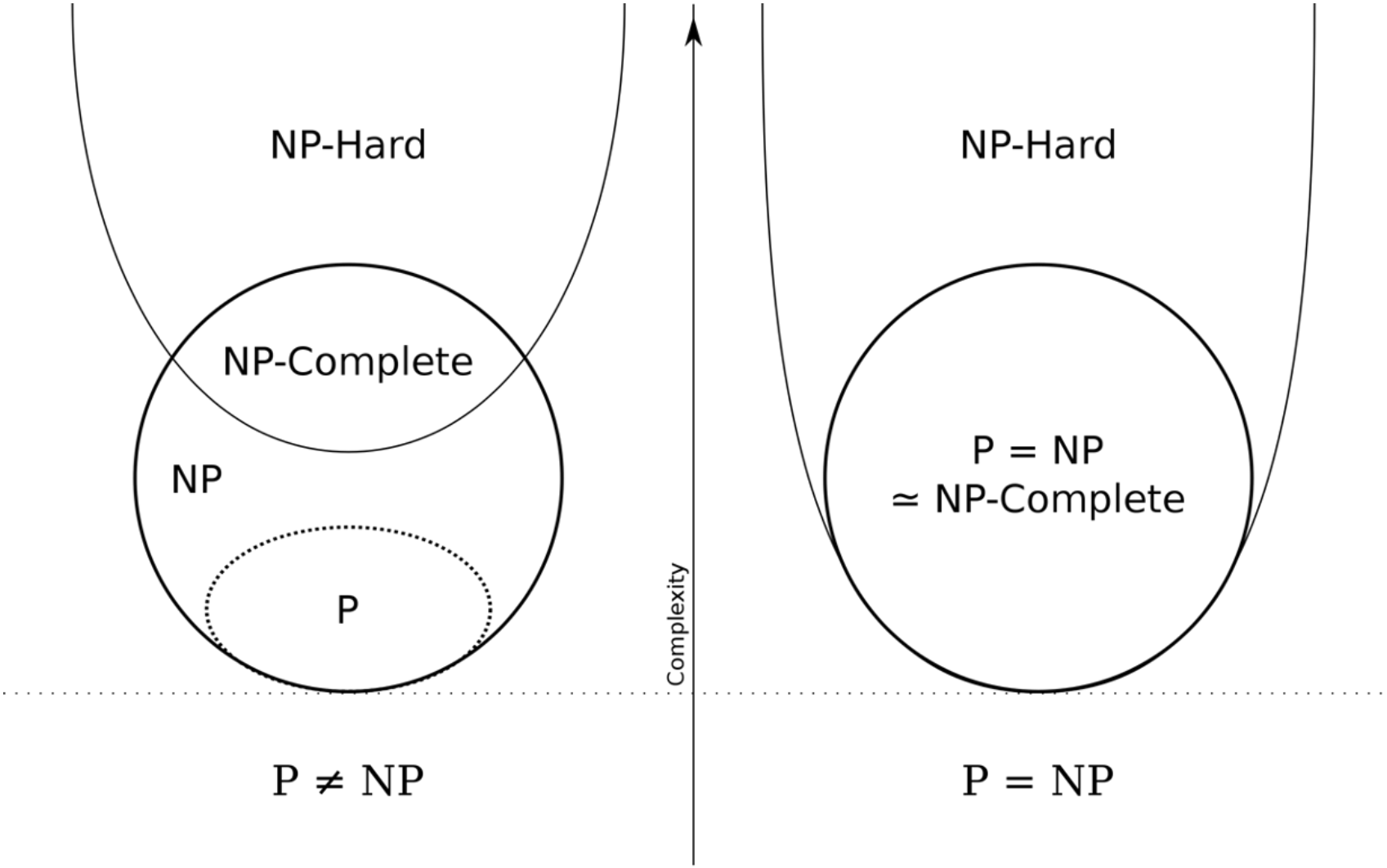
A problem L is **NP-hard** if some problem in NPC can be polynomially reduced to it (but L does not need to be in NP).

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

For example, recall

- VC: Given undirected graph G find a minimal size vertex cover.
- DVC: Given undirected graph G and k, is there a vertex cover of size k?

If we can solve the optimization problem **VC** we can easily solve the decision problem **DVC**. Simply run **VC** on graph G and find a minimal vertex cover S. Now, given (G, k), solve $DVC(G, k)$ by checking whether $k \geq |S|$. If $k \geq |S|$ answer Yes, if not, answer No.


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP
≃ NP-Complete
P ≠ NP
P = NP

Optional: "Intermediate" problems

These problems are in NP and believed to be outside P, yet decades of work have failed to show NP-completeness.

- Graph isomorphism (GI)
- Integer factorization

Exercise

Definition: Graph Isomorphism (GI)

Given graphs G_1 and G_2, is there a bijection of vertices that preserves adjacency?

Definition: Subgraph Isomorphism (SGI)

Given graphs H and G, does G contain a subgraph isomorphic to H?

Problem

Show that Subgraph Isomorphism (SGI) is NP-hard.

Hint

By reduction from k-Clique.