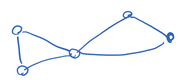
NP- Completeness

polynomial -time problem $O(n^k)$ ⟶ tractable problem

Super polynomial problem ⟶ intractable / hard problem

↳ $O(2^n)$

{ shortest-path problem : tractable
longest - path : intractable



{ Euler tour problem : visit every edge once , and you can visit vertices more than once ⟶ tractable

Hamiltian cycle : visit every vertex in graph exactly once
⟶ intractable

Complexity Classes

P : problems that can be decided in polynomial time

NP : " " " " verified " "

NP-hard : " " any NP problem can be reduced to in polynomial time

NP-complete : " " are both in NP and NP-hard

Decision Problem

A problem that the output to any input is either "Yes" or "No".

HAM-CYCLE : input , undirected graph $G = (V, E)$

Question (output) : Does G contains a cycle that visits every vertex exactly once?

PATH : inputs , $G = (V, E)$, pair of vertices $u, v \in V$, a number K.

Question : Is there a path in G from u to v with weight $\leq K$?

Question: Is there a path in G from u to v with weight ≤ K?

optimization / Search problem $\xrightarrow{\text{Convert}}$ decision problem

⇓

apply a bound on the optimization objective function

why focusing on decision problem?

- answer of decision problem is simple
- " " " " is unique (think of MST that didn't have unique answer)
- decision problem is at most as hard as the corresponding optimization problem

⟶ lower bound for decision problem will be
" " " optimization problem

Defining Problems as Languages:

Thinking of problem as a language
- input is a binary string
- output is either "Yes" or "No"

language of problem as set of input binary strings for which the correct output is "Yes"

$L_{HAM-CYCLE} = \{ G : G \text{ has a hamiltonian cycle} \}$

Deciding a language:

An algorithm decides a language if it correctly determines whether its input string is part of the language.

The algorithm:

- terminate for any input.
- return "Yes" if the input is in the language
- " "No" " " " not in the language

collection of decision problem instances

- Complexity Class P : (Polynomially Solvable)

- The set of problems that can be decided in polynomial time.

Formally, set of all languages L for which there exists an algorithm A and constant c:

- A decides L
- worst case run time of A is $O(n^c)$

Verification Algorithms

Inputs : - (binary) string x (the actual input)

- a certificate y (a proof that the correct answer for x is "Yes")

Output: either "Yes" or "No"

produce "Yes" if x belongs to L and y proves that.

Otherwise, produce "No".

language verified by a verification algorithm:

$$L = \{x : \text{there exists } y \text{ for which } A(x, y) \text{ produces "Yes"}\}$$

example: verification of HAM-CYCLE

inputs: ① an undirected graph $G = (V, E)$

② an ordered list of $\langle v_1, \dots, v_m \rangle$

Algorithm: produce "Yes" if

- sequence $\langle v_1, \ldots, v_m \rangle$ contain all vertices of V without any duplicate
- E contains edges (v_i, v_{i+1}) for $i = 1, \ldots, m-1$
- E contains edge (v_m, v_1)

otherwise, produce "No".

example: verification for PATH

inputs: - a weighted directed graph G, pair of vertices $u, v \in V$, a number K.

- an ordered list of vertices $\langle v_1, \ldots, v_m \rangle$

verification algorithm:
produce "yes" if:

- $v_1 = u$, $v_m = v$
- E contains edges (v_i, v_{i+1}) for $i = 1$ to $m-1$
- total weight of above edges is at most K

nondeterministic

Complexity Class NP: (polynomially verifiable)

Set of all problems for which "Yes" answer can be verified in poly time

Formally: Set of all languages L for which there exists verification algorithm A and a constant c:

- A verifies L
- worst-case run time of A is $O(n^c)$

Is $P \subseteq NP$? true

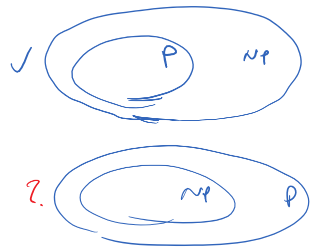
Consider verification algorithm that ignores the certificate input and polynomially solves the instance (since the problem is polynomially decidable)

Is $NP \subseteq P$? still unknown

if $NP \subseteq P$

$P \subseteq NP$

$\Longrightarrow P \overset{?}{=} NP$

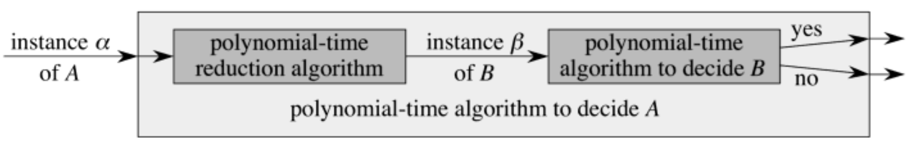


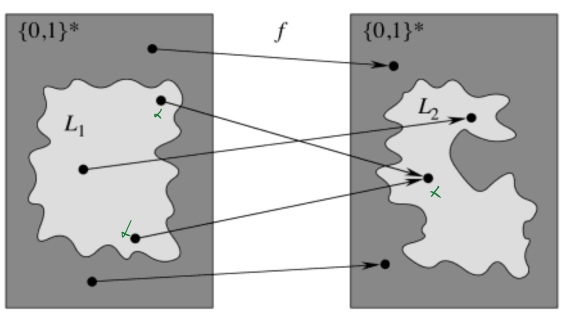
Reduction :

Language L_1 is polynomial-time reducible to another language L_2 if there exists a polynomial time algorithm f such that

$$x \in L_1 \iff f(x) \in L_2$$

$\boxed{\text{denoted by : } L_1 \leq_p L_2}$





What does $A \leq_p B$ mean ?

— A is no harder to solve than B

NP-hard complexity class :

Language L is NP-hard if for every $L' \in NP$, $L' \leq_p L$.

NP-Complete complexity class → hardest problems within NP.

Language that is both in NP and NP-hard.

How to prove a problem is NP-hard?

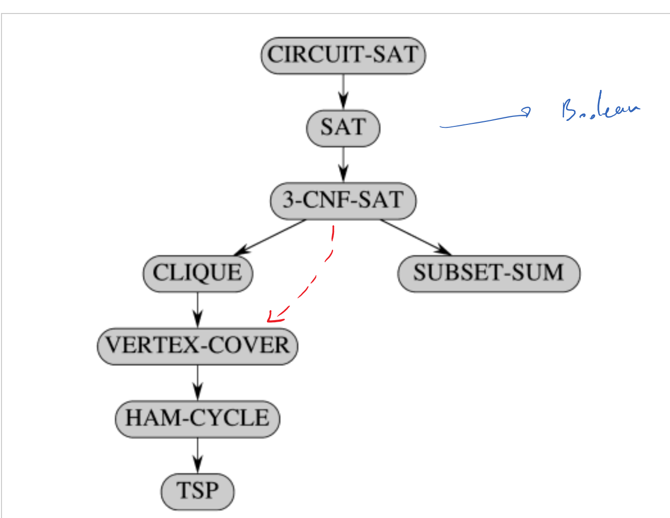
- find another problem L' that we know is NP-Complete
- describe (polynomial-time) algorithm to convert an instance of L' into instance of L
- show that algorithm is a reduction
  - for "Yes" instance of L', it should produce "Yes" instance of L
  - for "No" " " " " "No" " "
- show that algorithm is polynomial

$L' \leq_p L$ → $L \in$ NP-hard

L' is NP-complete → $L' \in NP$

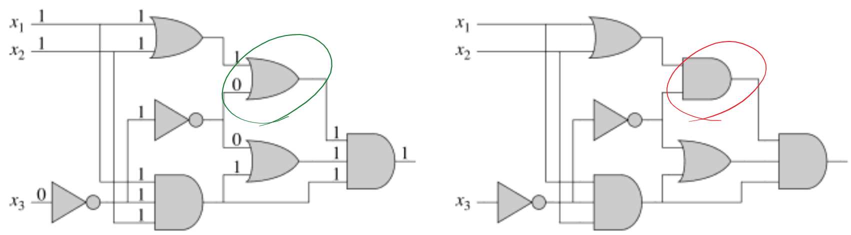
$L' \in$ NP-hard $\Longleftrightarrow \forall L'' \in NP,\ L'' \leq_p L'$

transitivity



→ Boolean satisfiability

CIRCUIT-SAT :



(a) (b)

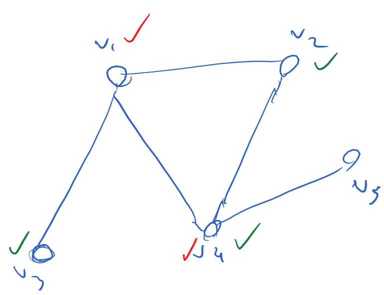
3-CNF-SAT :

CNF : Conjunctive Normal Form

- ↳ conjunction of clauses
- ↳ each clause is disjunction (OR) of 3 literals

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

Vertex Cover Problem



$V_0' = ✓$

✓ $V_1' = \{v_2, v_3, v_4\}$

✓ $V_2' = \{v_1, v_4\}$

VERTEX-COVER:

input : a graph G and integer k

question : is there a set of k vertices that are adjacent to all edges in G?

VERTEX-COVER is NP-Complete. Proof:

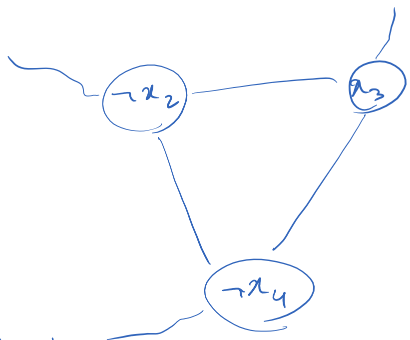
- VERTEX-COVER is in NP
  given a subset of vertices as certificate, we can verify if that's a cover in polynomial time.

- VERTEX-COVER is in NP-hard
  given an instance ϕ of 3-CNF-SAT with n variables and m clauses

form an instance of VERTEX-COVER (G, k):

- Truth-setting component: one vertex for each literal in ϕ,
  connecting x and $\neg x$

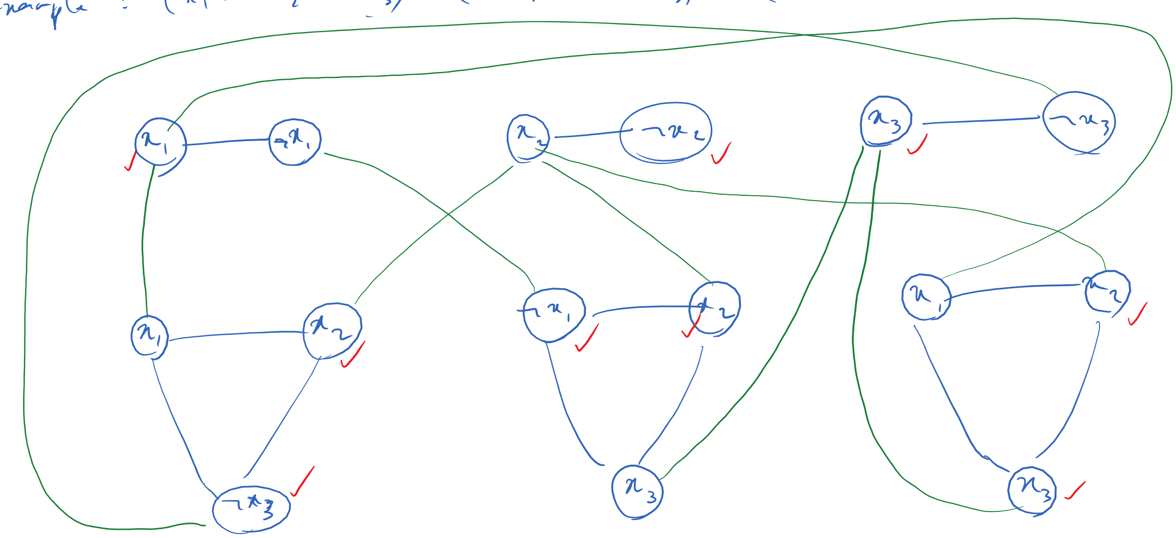
- clause-satisfaction components: three vertices corresponding to literals in each clause, connecting them together.



Also connect them each to corresponding vertex in truth-setting components.

Set $\underline{k = n + 2m}$

example: $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



Theorem ($\Rightarrow$): if ϕ is satisfiable, then G has vertex cover of size $n+2m$.

Given satisfying assignment t for ϕ, consider covering vertices:

- in truth-setting components, choose x_i if t sets it to true, or $\neg x$ otherwise.
- in clause-satisfaction components, find the first literal set to true, and choose the other two vertices

These are $n+2m$ vertices that cover all edges.

Theorem ($\Leftarrow$): if G has a vertex cover of size $n+2m$, then ϕ is satisfiable.

Suppose there exists cover A of G with size $n+2m$.

By construction:

- one vertex in truth-setting component
- two vertices in clause-satisfaction components

Let t be a truth assignment that makes x_i true iff its truth-setting vertex is in vertex cover A.

- For each clause C in ϕ, there is only one vertex not covered. Let's call that vertex v.

- A connecting edge connects v to a truth-setting vertex u.

  $v \notin A \implies u \in A$

- Therefore, literal corresponding to u is satisfied in t.

  $\implies C$ is satisfied

Since t satisfies each clause, it satisfies ϕ