NP-Completeness

- Most problems that we've seen are solvable in polynomial time

- Not all problems are. Some require exponential time, and some are impossible to solve (no matter how much time you give it. Ex: halting problem)

- There are also problems that seem to require exponential time, but nobody has been able to prove that no polynomial time algorithm exists for solving them.

Decision vs Optimization problems

- In an optimization problem we want to maximize or minimize some objective.

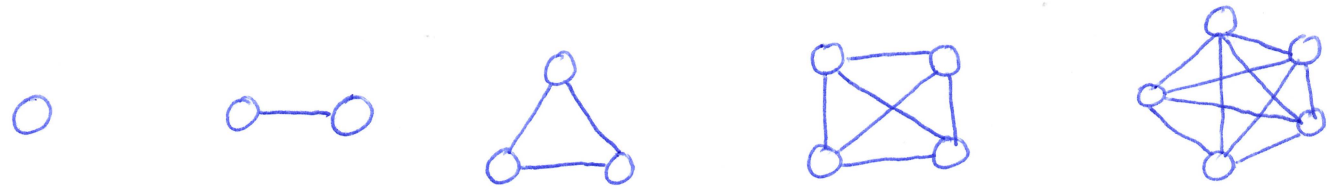
  Example: Given two strings X and Y, find the longest common subsequence between them.

- A decision problem is a special kind of problem whose answer is either YES or NO.

  Example: Given two strings X and Y, and an integer k, is there a common subsequence between them with length at least k?

Clique

A clique is a complete subgraph of an undirected graph.



clique of size 1 | size 2 | size 3 | size 4 | size 5

Optimization problem

- Given an undirected graph G, what's the size of its largest clique?

Decision problem

- Given an undirected graph G and an integer k, is there a clique of at least k nodes?

- Optimization problem can be solved by iteratively solving the decision problem for $k = 1, 2, 3, \ldots$

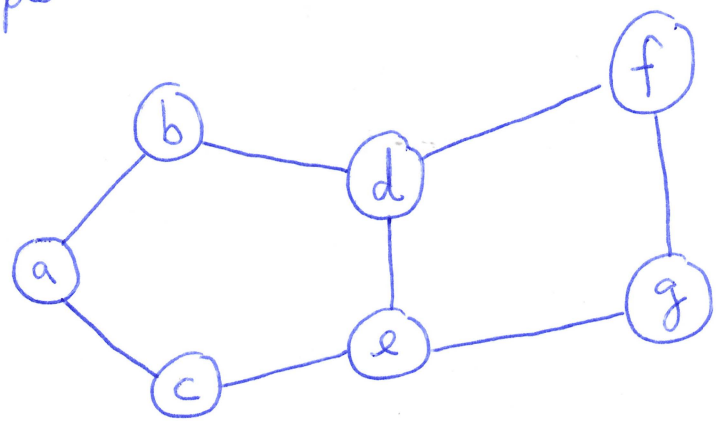
- One way of solving the decision version of the Clique problem.
  - try all $\binom{n}{k}$ combinations of nodes of a graph, and check if we get a clique or not.
  - Running time: $\theta(n^k)$

    Polynomial? Not really because k can be $\theta(n)$

Independent Set

- Given an undirected graph G, we say that a subset of vertices S is an <u>independent set</u> if there are no edges between any two nodes in S.

- Example



$\{a, d, g\}$ is an independent set.

Opt. problem : Given an undirected graph G find

- an independent set whose size is as large as possible.

Decision problem : Given an undirected graph G and an integer k, is there an independent set with at least k nodes?

Application example

- vertices → courses
- edge (u, v) means course u and v have at least one student in common
- Independent Set → set of courses whose final exam could be given at the same time.

Graph coloring

- An undirected graph is k-colorable if we can assign one of the k colors to each node in such a way that no edge have their ends with the same color.

Optimization problem

- Given an undirected graph find the smallest k that makes the graph k-colorable

Decision problem

- Given an undirected graph and an integer k, is the graph k-colorable?

- From now on we restrict ourselves to decision problems (by iterating over k we can use the decision problem to solve the optimization version of it.)

- It seems hard to find solutions to many problems (Clique, Independent Set, k-colorability, are some examples)

- But many times it's <u>easy to verify</u> if a solution is correct or not.

  - Clique → check that each of the $\binom{k}{2}$ edges among the k nodes really exist.

  - Independent Set → check that there's no edge between any two nodes of the set.

  - k-colorability → check every edge of the graph and confirm that the colors of the endpoints are different.

Classes of Problems

P → Decision problems that can be solved in polynomial time.

NP → Decision problems that can be verified in polynomial time

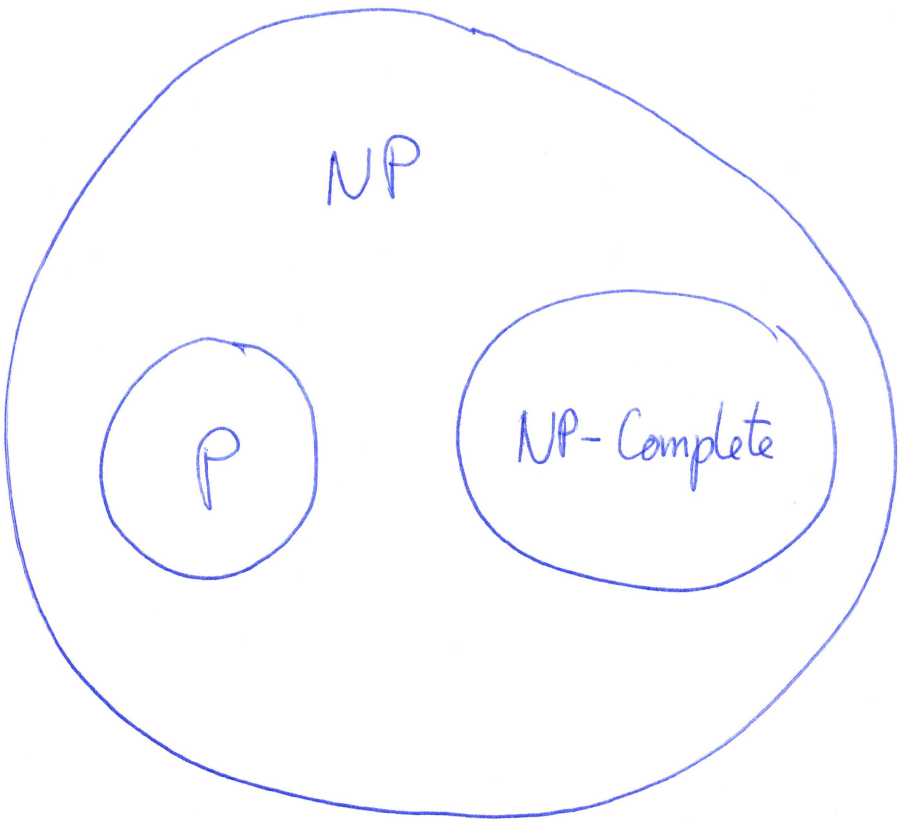
(NP stands for nondeterministic polynomial which informally means the ability to correctly guess a solution.)

NP-Complete → Problems that are in NP and that are "as hard as" any other problem in NP

- Most people believe $P \neq NP$ but nobody has been able to prove it

- If one could find a polynomial time algorithm for an NP-Complete problem, then all of the NP-Complete problems would also be solvable in polynomial time.

$\Rightarrow$ All the NP-Complete problems are in some sense "equivalent"



To show that a problem X is NP-Complete, we need to show 2 things.

1) X is in NP

2) Pick a problem Y known to be NP-Complete and show that,

$$Y \leq_P X$$

means Y is polynomial-time reducible to X

- A polynomial-time reduction is a procedure that transforms an arbitrary instance of Y into an instance of X in such a way that,
  - the procedure takes polynomial-time
  - a YES answer to the X instance implies a YES answer to the original Y instance, and vice-versa.

- If $Y \leq_p X$, then we can say that X is as hard as Y.

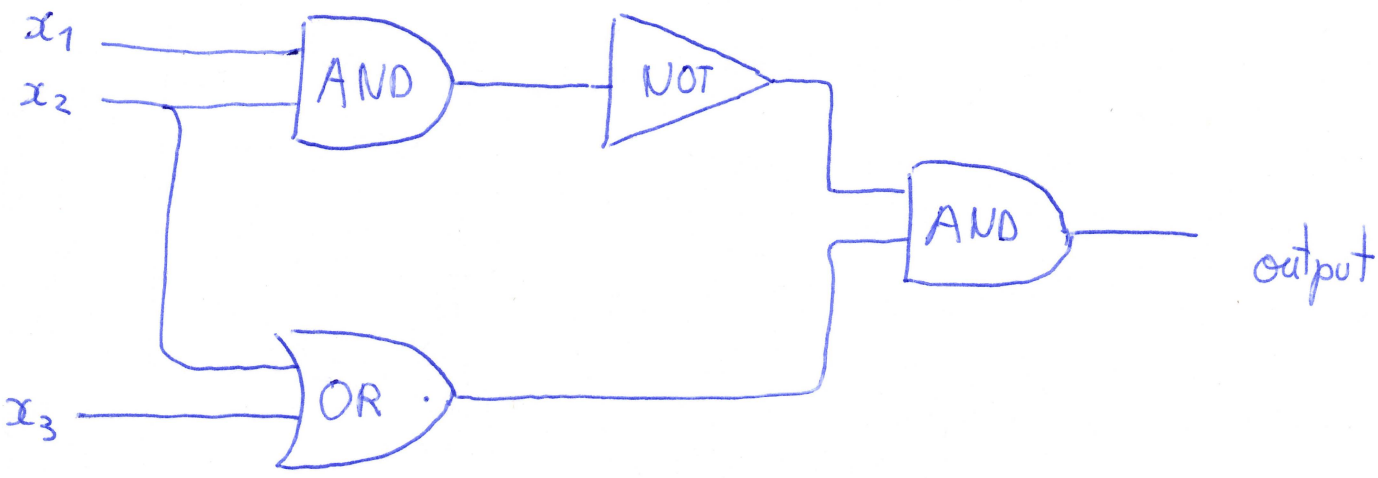
- If we can solve X, we can also solve Y.
  → all we need is a little overhead for the transformation (polynomial-time)

- Reductions are transitive

  If $X \leq_p Y$ and $Y \leq_p Z$,

  then $X \leq_p Z$

- Need a first NP-Complete problem
- Cook and Levin (early 70s) proved that CIRCUIT-SAT is NP-Complete.
  - Circuit Satisfiability
    - Given a combinatorial circuit built from AND, OR, and NOT gates, is it possible to have an assignment to the circuit inputs such that the circuit outputs 1?



3-SAT

- Given a formula in conjunctive normal form (CNF) with exactly 3 literals per clause, is the formula satisfiable?

- Example:

$$(\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_4 \vee x_1 \vee x_3)$$

Formula is satisfiable by setting $\begin{cases} x_1 = 0 \\ x_2 = 0 \\ x_3 = 1 \\ x_4 = 0 \end{cases}$

- A formula is satisfiable if there's an assigment to the input variables that makes the formula evaluate to 1.

- CNF has the form $C_1 \wedge C_2 \wedge \dots \wedge C_m$ where ~~each~~ each C_i is a clause (the OR of variables or their negations)

Let's show that 3-SAT is NP-Complete

1) 3-SAT is in NP

Given an assignment to variables, just check if it evaluates to 1. Clearly takes polynomial time in the number of variables.

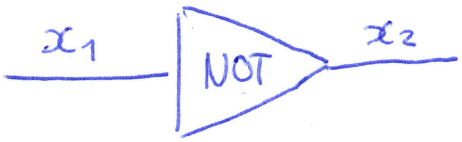
2) CIRCUIT-SAT $\leq_p$ 3-SAT

Take an arbitrary instance of CIRCUIT-SAT.

- create a variable x_i for each wire of the circuit.
- for each gate in the circuit create a CNF formula in such a way that the formula behaves according to the rules of the gate. (3 gate types: AND, OR, NOT)
- hard code fixed inputs
- hard code output variable to be 1
- create a larger CNF by ANDing the CNF formulas corresponding to each gate + fixed input + output
- make all clauses have exactly 3 literals.

Handling the 3 gate types

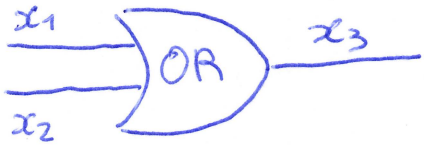
NOT gates



$$x_2 = \bar{x}_1$$

$$\Leftrightarrow (x_2 \vee x_1) \wedge (\bar{x}_2 \vee \bar{x}_1)$$

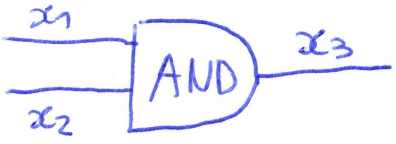
OR gates



$$x_3 = x_1 \vee x_2$$

$$\Leftrightarrow (\bar{x}_3 \vee x_1 \vee x_2) \wedge (x_3 \vee \bar{x}_1) \wedge (x_3 \vee \bar{x}_2)$$

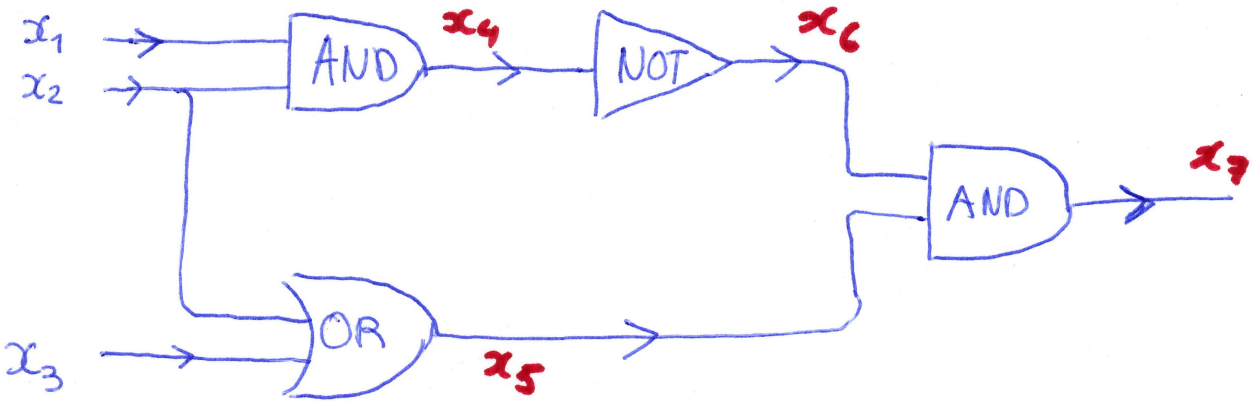
AND gates



$$x_3 = x_1 \wedge x_2$$

$$\Leftrightarrow (x_3 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_3 \vee x_1) \wedge (\bar{x}_3 \vee x_2)$$

Example



$x_4 = x_1$ AND x_2 ⟶ $(x_4 \vee \bar{x}_1 \vee \bar{x}_2)$ $(\bar{x}_4 \vee x_1)$ $(\bar{x}_4 \vee x_2)$

$x_5 = x_2$ OR x_3 ⟶ $(\bar{x}_5 \vee x_2 \vee x_3)$ $(x_5 \vee \bar{x}_2)$ $(x_5 \vee \bar{x}_3)$

$x_6 =$ NOT x_4 ⟶ $(x_6 \vee x_4)$ $(\bar{x}_6 \vee \bar{x}_4)$

$x_7 = x_6$ AND x_5 ⟶ $(x_7 \vee \bar{x}_6 \vee \bar{x}_5)$ $(\bar{x}_7 \vee x_6)$ $(\bar{x}_7 \vee x_5)$

output ⟶ (x_7)

Making all clauses have 3 literals

- Add temporary variables.
- Clauses with 1 variable:

$$(x_7) \iff (x_7 \vee a \vee b)$$
$$(x_7 \vee \bar{a} \vee b)$$
$$(x_7 \vee a \vee \bar{b})$$
$$(x_7 \vee \bar{a} \vee \bar{b})$$

- Clauses with 2 variables:

$$(\bar{x}_7 \vee x_5) \iff (\bar{x}_7 \vee x_5 \vee c)$$
$$(\bar{x}_7 \vee x_5 \vee \bar{c})$$

- Reduction takes polynomial-time.
- The resulting 3-CNF formula is satisfiable if and only if the original circuit is satisfiable.

  Why? because the clauses were created to match the behavior of the logic gates.

- We just proved that 3-SAT is NP-Complete.

- If we could solve 3-SAT in polynomial time, then we would be able to solve CIRCUIT-SAT in polynomial time as well.

- Why?

  Because we can transform (in polynomial time) any instance of CIRCUIT-SAT into an instance of 3-SAT, and the 3-SAT instance will produce a YES answer if and only if the original circuit produces a YES answer.

Show that Independent Set is NP-Complete

1) Indep. Set is in NP.
(check that there's no edge between any two nodes of the set can be done in polynomial time)

2) 3-SAT $\leq_p$ Indep. Set

i) Need to choose one term from each clause, and find an assignment that causes all clauses to evaluate to 1.

ii) Can do it as long as there's no "conflicts"
(selecting x_i and $\bar{x}_i$ is a conflict)

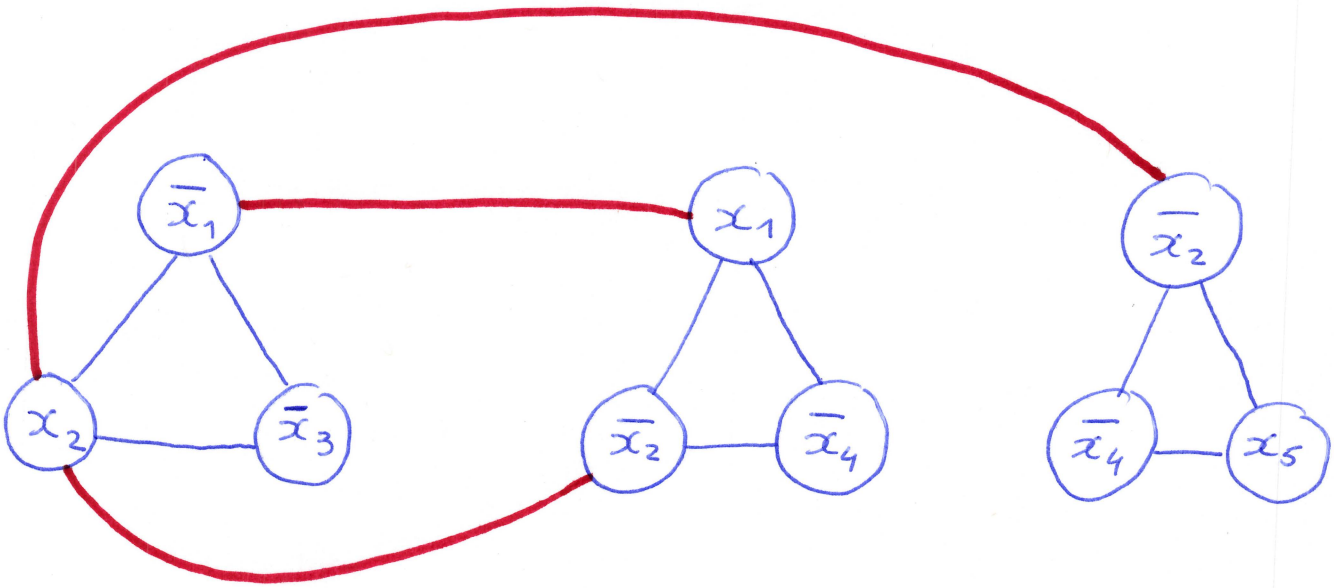
Procedure to transform an arbitrary instance of 3-SAT into an instance of Independent Set.

a) Suppose the 3-SAT instance has m clauses.
Create a graph G with 3m nodes grouped into m triangles, one for each clause.

b) Add edges between x_i and $\bar{x}_i$ to prevent conflicts.

Example

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_5)$$



- An independent set of size m would consist of picking one node from each of the triangles.
  (cannot choose 2 nodes from the same triangle. Why?)
- Red edges take care of conflicts.
- After doing this transformation, the original 3-SAT instance is satisfiable if and only if the graph G has an independent set of m nodes.

1) 3-SAT = YES $\Rightarrow$ Indep Set of size m = YES

2) Indep Set of size m = YES $\Rightarrow$ 3-SAT = YES

- The procedure to transform an instance of 3-SAT into an instance of Indep. Set is clearly polynomial.

Bottom line

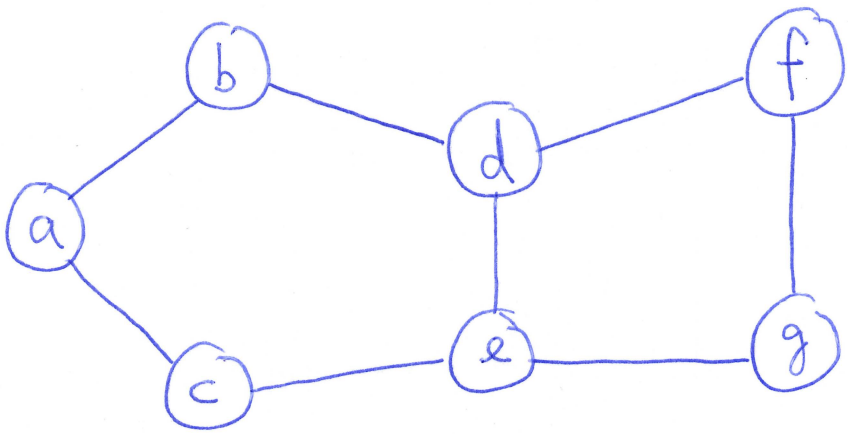
- If we were able to solve Indep. Set in polynomial time, we would also be able to solve 3-SAT in polynomial time.

$\Rightarrow$ Indep. Set is as hard as 3-SAT.

Vertex Cover

Given a graph $G = (V, E)$, a set of nodes $S \subseteq V$ is a vertex cover if every edge of the graph has at least one end in S.

Vertex Cover Problem : Given a graph G and an integer k, does G contain a vertex cover of size at most k?



$\{b, c, e, f\}$ is a vertex cover

Vertex Cover is NP-Complete

1) Vertex Cover is in NP
   - just go over all the edges of the graph and check that one of the ends is in the set. Can be done in polynomial time.

2) Indep. Set $\leq_p$ Vertex Cover

> Let $G=(V,E)$. S is an independent set if and only if its complement $V-S$ is a vertex cover.

Proof.

- Suppose S is an independent set. Pick an edge (u,v) from the graph. u and v cannot be both in S (otherwise S wouldn't be an independent set). Therefore, at least one of them has to belong to $V-S$. This happens for every edge of the graph. Therefore $V-S$ is a vertex cover.

- Likewise, suppose $V-S$ is a vertex cover. Consider any two nodes u and v in S. If the graph had an edge (u,v) then both ends would not be in $V-S$ (contradicting the fact that $V-S$ is a vertex cover). Therefore (u,v) cannot be an edge, and we can conclude that S is an independent set.

Indep. Set $\leq_p$ Vertex Cover

- If we have an algorithm to solve Vertex Cover, then we can decide if a graph $G=(V,E)$ has an independent set of size k by asking the algorithm to find if G has a vertex cover of size at most $|V|-k$.