NP-Hardness

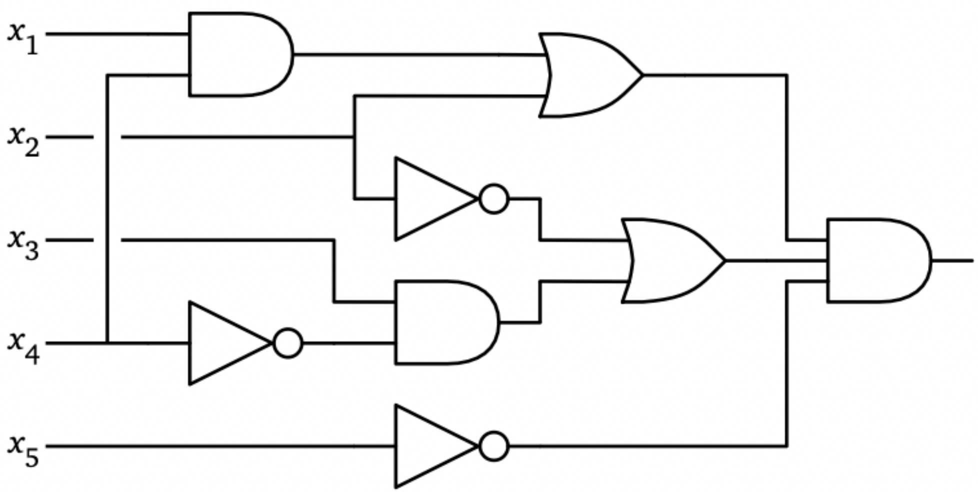
References:

- Algorithms, Jeff Erickson, Chapter 12
- The Algorithm Design Manual, Chapter 9

CircuitSat

Here's a simple looking problem: **circuit satisfiability.**

Given a boolean circuit (consist of inputs and an output, AND, OR, NOT gates), determine whether there's a set of inputs that make the circuit output TRUE, or there isn't such inputs.



It is not too hard to solve the presented problem in figure, but how to solve it **in general**, for any given circuits?

For a given set of inputs $\{x_1, \ldots, x_n\}$ it's easy to compute the output (how?). We can test all the possible inputs (in total 2^n distinct ones) and see if one of them can make output TRUE.

It turns out that noboby has come up a way of solving this problem faster than basically trying all 2^n possible inputs, which would be exponential time complexity.

This is a **hard** problem!

P vs. NP

Efficient algorithm: polynomial time $O(n^c)$, where c is a constant, and n is the problem size.

Problems like the CircuitSat make people think what kind of problems admit efficient algorithm.

We consider decision problems, whose outputs are boolean yes/no. We classify them into three categories:

- **P**: easy problem; can be solved in polynomial time
- **NP**: maybe hard, but if the answer is YES, then there is a **proof** of this fact that can be **checked** in polynomial time. E.g. CircuitSat.

- **co-NP**: maybe hard, we can check NO answer in polynomial time.

CircuitSat belongs to **NP**. It's widely believed that it does not belong to **P** (but it's an open question).

From the definition, P$\subseteq$NP, and also P$\subseteq$co-NP.

> P vs. NP problem: are P and NP really different?

If you can prove this, you can claim $1,000,000 from the Clay Mathematics Institute.

Question 1: Why only consider decision problems though?

Most algorithms can be phrased as decision problems which captures the essence of the computation. Example:

The Max Independent Set (MaxIndSet) problem: Given a graph, what's the size of largest independent set of nodes? Independent set of nodes do not have edges between them.

It can be rephrased as decision problem:

Given a graph G and integer k, does there exist an independent set of size k?

If we can solve this yes/no problem efficiently, we can use binary search to find the optimal solution to general TSP.

Question 2: Why study NP-completeness?

Well, suppose your boss asks you to solve a problem but you fail to find a fast algorithm.

What can you say?

1. I am dumb... (your job might be in danger)
2. There is NO fast algorithm! (How do you know? Do you have a lower bound?)
3. I can't solve it, but nobody else can either... (by NP-completeness reduction, intellectual gynmastics).

Obviously, answer 3 makes you look smarter. We will study how to do NP-completeness reduction, which will help us spot and prove hardness of algorithmic problems.

NP-hard/NP-complete

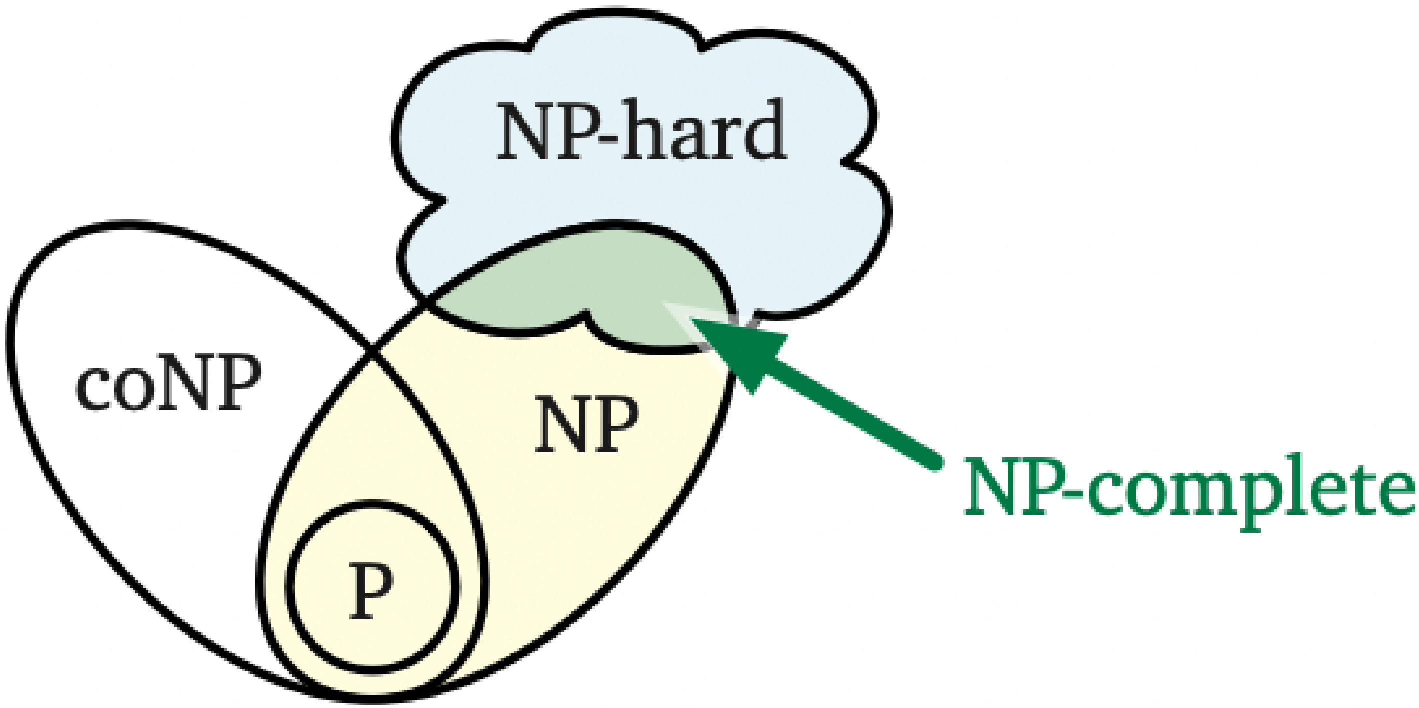
A problem Π is **NP-hard** if a polynomial algorithm of Π would imply a polynomial time algorithm for *every* problem in NP.

Π is NP-hard $\Longleftrightarrow$ if Π can be solved in polyn time, then P=NP.

By this definition, NP-hard problems are at least as hard as NP.

If a problem is **NP-complete**, if it's both NP-hard and NP. So NP-complete problems are the hardest in NP.

(we could probably call NP by NP-easy, in contrast to NP-hard).



This is how we *think* they look like.

There are thousands of problems in the NP-complete set. Solving one of them implies solving all of them. So far not a single one is solved, which gives strong suggestion that NP≠P.

NP-hard problem are pretty strong; do we have *any* problem that is NP-hard?

Well, we just discussed one: CircuitSat is NP-hard. This is a theorem by Cook in 1971 and Levin in 1973, which is by no means trivial to prove.

The Cook-Levin Theorem: Circuit satisfiability is NP-hard.

Reductions and SAT

We have one problem that is shown to be NP-hard so far.

From there, showing other problems to be NP-hard is much easier:

> Just reduce one of the NP-hard problem (e.g., CircuitSat) to your problem. (not the other way around!)

(alternatively, ask yourself, if I can solve my problem efficiently, can it be used to solve on of the NP-hard problems efficiently?)

This technique is called **reduction**.

As an example, let's say we face a new problem: **formula satisfiability** problem (SAT). The input to SAT is a boolean formula like:

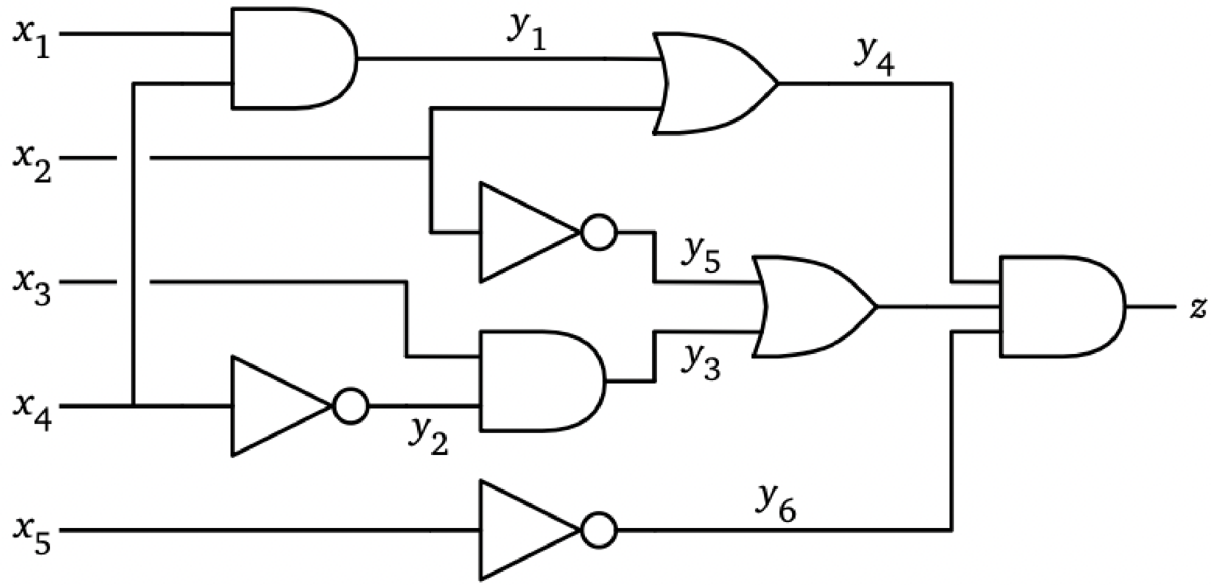
$$a \vee b \vee c \vee \bar{d} \Leftrightarrow ((b \vee \bar{c}) \vee \overline{(\bar{a} \Rightarrow d) \vee (c \neq a \vee b)})$$

the question is: isthere a set of boolean values for the variables to make the formula TRUE.

To prove that SAT is NP-hard, we reduce one NP-hard problem to it. We only know one NP-hard problem, that is CircuitSat.

So our plan is: suppose we can solve arbitrary SAT problem efficiently. How can we solve CircuitSat efficiently, using our magic, fast SAT solver, as black-box/subroutine?

We start with an arbitrary boolean circuit. We turn the circuit K into a boolean formula Φ as follows:



$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge$$
$$(y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

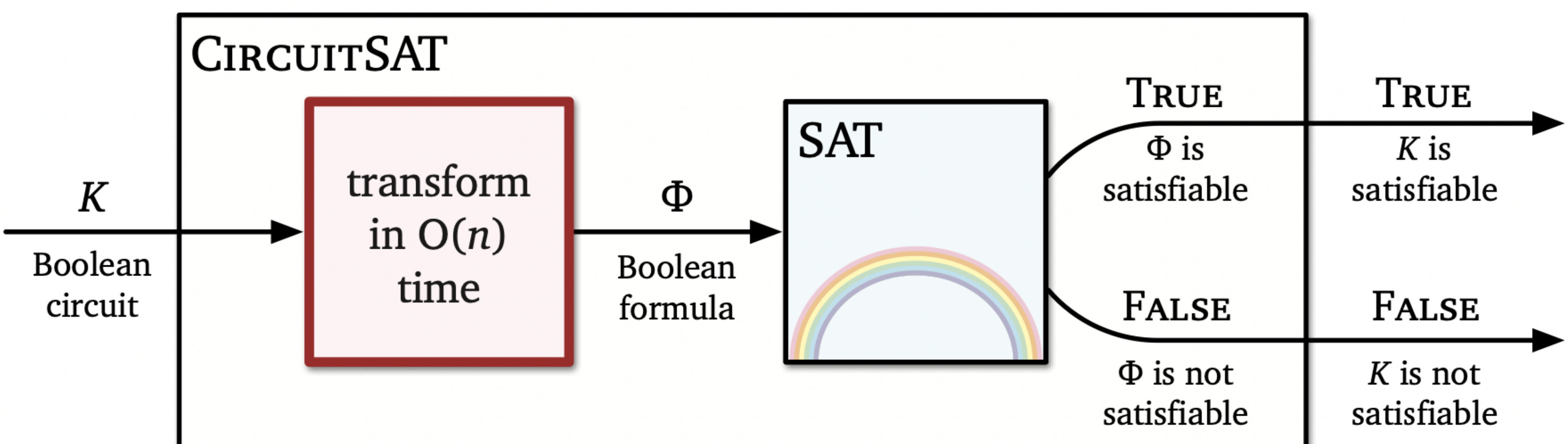
Each inner wire is assigned a variable y_i, output is assigned z. Each gate becomes an equation. Then AND them altogether.

Claim: the circuit is satisfiable iff the boolean formula is satisfiable.

=>: Given a set of inputs that satisfy the circuit, assign variables according the gates. The formula satisfies.

<=: given a satisfied formula, ignoring y_i and z. x_i are the inputs to the circuit that satisfies. □

Additionally we must prove the transformation of the problem is efficient: it can be done in $O(n)$ time, and the resulting problem has (essentially) the same input size (up to constant time larger).



3SAT

A special case of SAT which is very useful in proving NP-hardness is called 3SAT.

A boolean formula is *conjunctive normal form* (CNF) if it's conjunction (and) of several *clauses*, each of which is the disjunction (or) of several *literals*, each of which is either a variable of a negated variable.

$$\overbrace{(a \vee b \vee c)}^{\text{3-literal clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (\bar{b} \vee c \vee \bar{d})$$

If each clause has exactly 3 literals, then it's call 3CNF. This appears to a special case of SAT problem. Is it actually easier?

It turns our it's as hard as SAT (they are equivalent).

It's easier to prove that 3SAT problem is NP-hard, by again, reducing CircuitSat problem to 3SAT.

Plan: reducing arbitrary circuit K into equivalent 3CNF formulas! Four steps:

1. **Make sure every AND and OR gate in K has exactly two inputs**. If not, replacing multi-input AND/OR gate with binary tree of more gates. $K \rightarrow K'$
2. **Transcribe K' into boolean formulat Φ_1 with one clause per gate**: (the same as CircuitSat->SAT)
3. **Replace each clause in Φ_1 with a CNF formula:** $\Phi_1 -> \Phi_2$

$$a = b \wedge c \quad \rightarrow \quad (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee c)$$

$$a = b \vee c \quad \rightarrow \quad (\bar{a} \vee b \vee c) \wedge (a \vee \bar{b}) \wedge (a \vee \bar{c})$$

$$a = \bar{b} \quad \rightarrow \quad (a \vee b) \wedge (\bar{a} \vee \bar{b})$$

4. Replace each clause in Φ_2 with a 3CNF formula Φ_3

$$a \wedge b \;\rightarrow\; (a \vee b \vee x) \wedge (a \vee b \vee \bar{x})$$

$$z \;\rightarrow\; (z \vee x \vee y) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee x \vee \bar{y}) \wedge (z \vee \bar{x} \vee \bar{y})$$

Φ_1 is equivalent to Φ_2. Every assignment that satisfies Φ_2 will also satisfy Φ_3, by assigning arbitrary x,y,z. Conversely, every assignment that satisfies Φ_3 also satisfies Φ_2, by ignoring x,y,z.

The problem Φ_3 is only constant factor larger than κ_1, and the transformation ca be done in linear time (polynomial time would be enough). Thus we proved 3SAT is NP-hard. □

Example: Reducing to 3CNF-SAT problem (the previous CircuitSat example). Looks much larger, but it's only constant factor larger.

$$
\begin{gathered}
(y_1 \vee \overline{x_1} \vee \overline{x_4}) \wedge (\overline{y_1} \vee x_1 \vee z_1) \wedge (\overline{y_1} \vee x_1 \vee \overline{z_1}) \wedge (\overline{y_1} \vee x_4 \vee z_2) \wedge (\overline{y_1} \vee x_4 \vee \overline{z_2}) \\
\wedge (y_2 \vee x_4 \vee z_3) \wedge (y_2 \vee x_4 \vee \overline{z_3}) \wedge (\overline{y_2} \vee \overline{x_4} \vee z_4) \wedge (\overline{y_2} \vee \overline{x_4} \vee \overline{z_4}) \\
\wedge (y_3 \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{y_3} \vee x_3 \vee z_5) \wedge (\overline{y_3} \vee x_3 \vee \overline{z_5}) \wedge (\overline{y_3} \vee y_2 \vee z_6) \wedge (\overline{y_3} \vee y_2 \vee \overline{z_6}) \\
\wedge (\overline{y_4} \vee y_1 \vee x_2) \wedge (y_4 \vee \overline{x_2} \vee z_7) \wedge (y_4 \vee \overline{x_2} \vee \overline{z_7}) \wedge (y_4 \vee \overline{y_1} \vee z_8) \wedge (y_4 \vee \overline{y_1} \vee \overline{z_8}) \\
\wedge (y_5 \vee x_2 \vee z_9) \wedge (y_5 \vee x_2 \vee \overline{z_9}) \wedge (\overline{y_5} \vee \overline{x_2} \vee z_{10}) \wedge (\overline{y_5} \vee \overline{x_2} \vee \overline{z_{10}}) \\
\wedge (y_6 \vee x_5 \vee z_{11}) \wedge (y_6 \vee x_5 \vee \overline{z_{11}}) \wedge (\overline{y_6} \vee \overline{x_5} \vee z_{12}) \wedge (\overline{y_6} \vee \overline{x_5} \vee \overline{z_{12}}) \\
\wedge (\overline{y_7} \vee y_3 \vee y_5) \wedge (y_7 \vee \overline{y_3} \vee z_{13}) \wedge (y_7 \vee \overline{y_3} \vee \overline{z_{13}}) \wedge (y_7 \vee \overline{y_5} \vee z_{14}) \wedge (y_7 \vee \overline{y_5} \vee \overline{z_{14}}) \\
\wedge (y_8 \vee \overline{y_4} \vee \overline{y_7}) \wedge (\overline{y_8} \vee y_4 \vee z_{15}) \wedge (\overline{y_8} \vee y_4 \vee \overline{z_{15}}) \wedge (\overline{y_8} \vee y_7 \vee z_{16}) \wedge (\overline{y_8} \vee y_7 \vee \overline{z_{16}}) \\
\wedge (y_9 \vee \overline{y_8} \vee \overline{y_6}) \wedge (\overline{y_9} \vee y_8 \vee z_{17}) \wedge (\overline{y_9} \vee y_6 \vee z_{18}) \wedge (\overline{y_9} \vee y_6 \vee \overline{z_{18}}) \wedge (\overline{y_9} \vee y_8 \vee \overline{z_{17}}) \\
\wedge (y_9 \vee z_{19} \vee z_{20}) \wedge (y_9 \vee \overline{z_{19}} \vee z_{20}) \wedge (y_9 \vee z_{19} \vee \overline{z_{20}}) \wedge (y_9 \vee \overline{z_{19}} \vee \overline{z_{20}})
\end{gathered}
$$

Reduction: Genereal Pattern

All NP-Hardness proofs — polynomial time reductions — follow the same general outline:

1. Describe a polynomial time algorithm to transform an **arbitrary** instance of x of X into a special instance y of Y.
2. Prove that if x is "good" instance of X, then y is "good" instance of Y
3. Prove that if y is "good" instance of Y, then x is "good" instance of X (!)

What does "good" mean? It means there's certificate. E.g.

To reduce problem X to Y, we actually need to design 3 algorithms:

1. Transform arbitrary instance x of X to a special instance y of Y in polynomial time.
2. Transform an arbitrary certificate for x into certificate of y.
3. Transform an arbitrary certificate for y into certificate of x.

Notes:

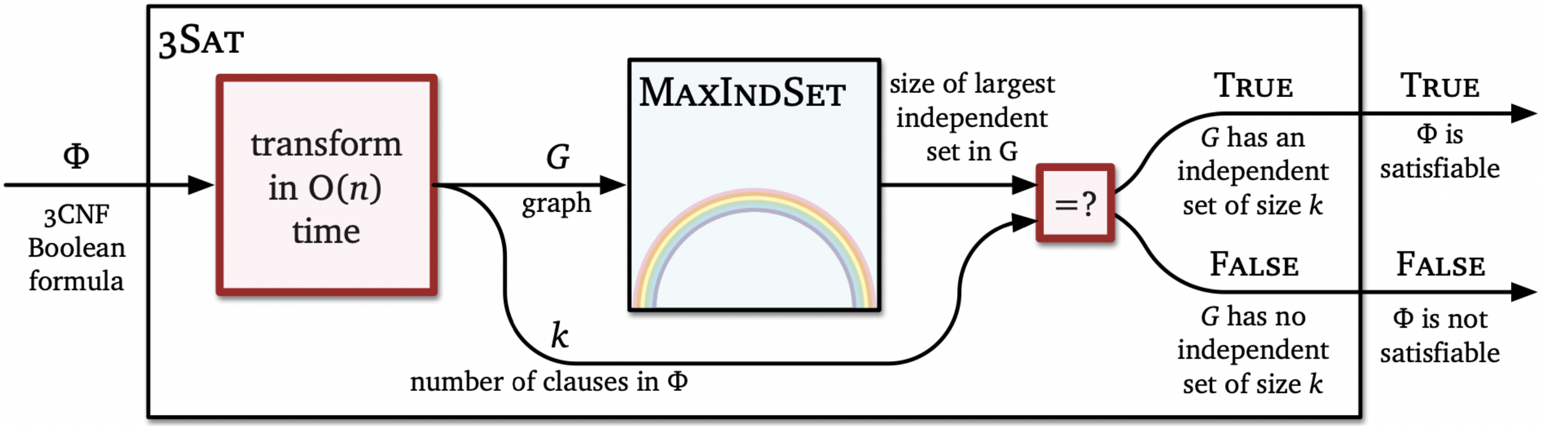
- Asymmetry: only convert x to y; **not** y to x. Key point! We need not think about arbitrary y of Y, only the speical y (probably highly structured)!
- Symmetry: we must convert certificates from x to y and from y to x.

3SAT->Maximum Independence Set

Given a undirected graph. An ***independent set*** is a subset of the nodes with no edges between them.

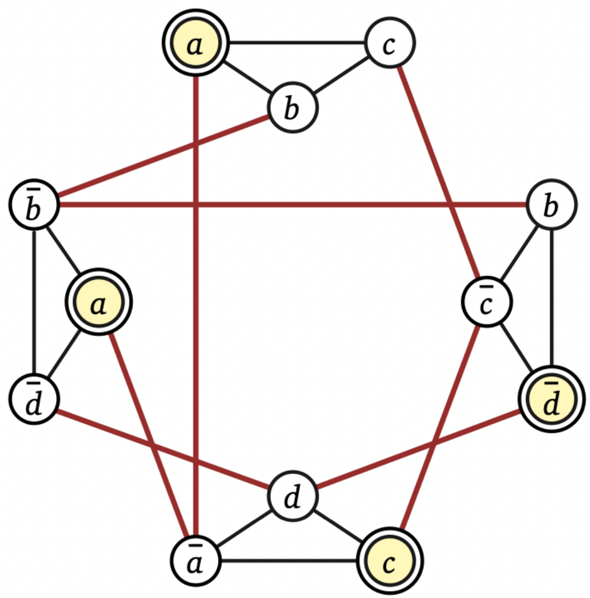
Maximum Independent Set (**MaxIndSet**) problem is to find the size of largest independent set.

We prove MaxIndSet is NP-hard by reducing 3SAT to MaxIndSet. Plan:



Suppose we can solve MaxIndSet. Can we solve 3SAT?

From any 3SAT problem instance, we can construct a graph like this:



$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

1. Every variable in every clause is a node.
2. Edges between nodes iff a) they are in the same clause; b) they are the negation of the same variable.

It's clear that the graph has MaxIndSet of size at most k (the number of clauses).

Further we claim: the graph has MaxIndSet of size exactly k, if and only if the original formula Φ is satifiable.

- (=>) Suppose Φ is satifiable. Fix any satisfiable assignment. Every clause must have at 1 TRUE literal. We can then pick 1 node corresponding to the TRUE literal from each triangle. Is there any edge between the nodes we pick?

- (<=) Suppose G contains Independent Set S of size k. Each node in the independent set must be in different triangle. Suppose we assign TRUE to the literal corresponding to the node in S (why is this assignment consistent? Because contradicting literals are connected by edges). S must contain one node in every triangle (clause). There each clause is TRUE. The formula Φ is therefore satisfiable.

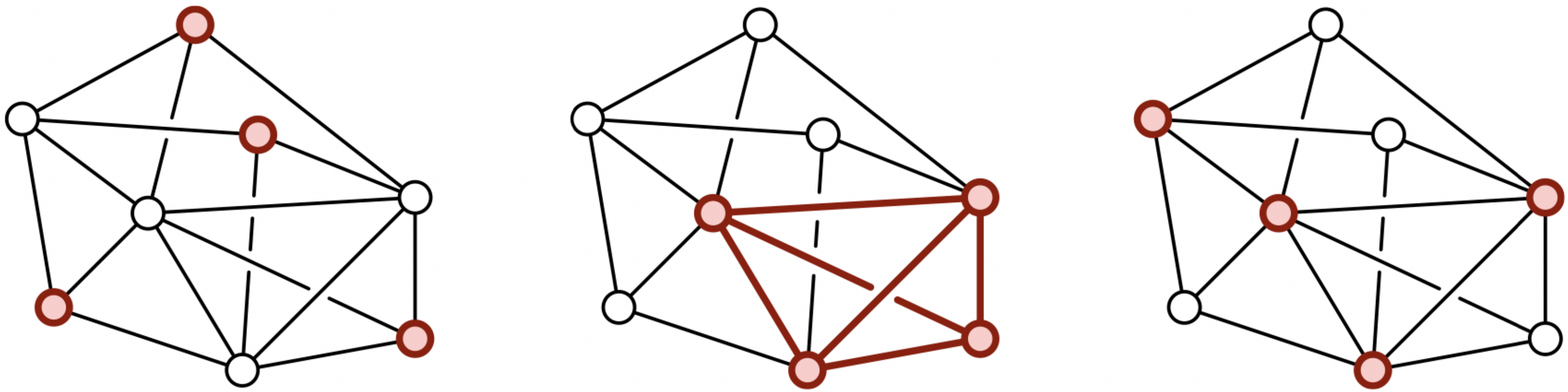
Transforming 3SAT formula Φ to the graph G costs polynomial time.

MaxIndSet->MaxClique

A ***clique***, aka complete graph, is a graph where every pair of nodes is connected by an edge.

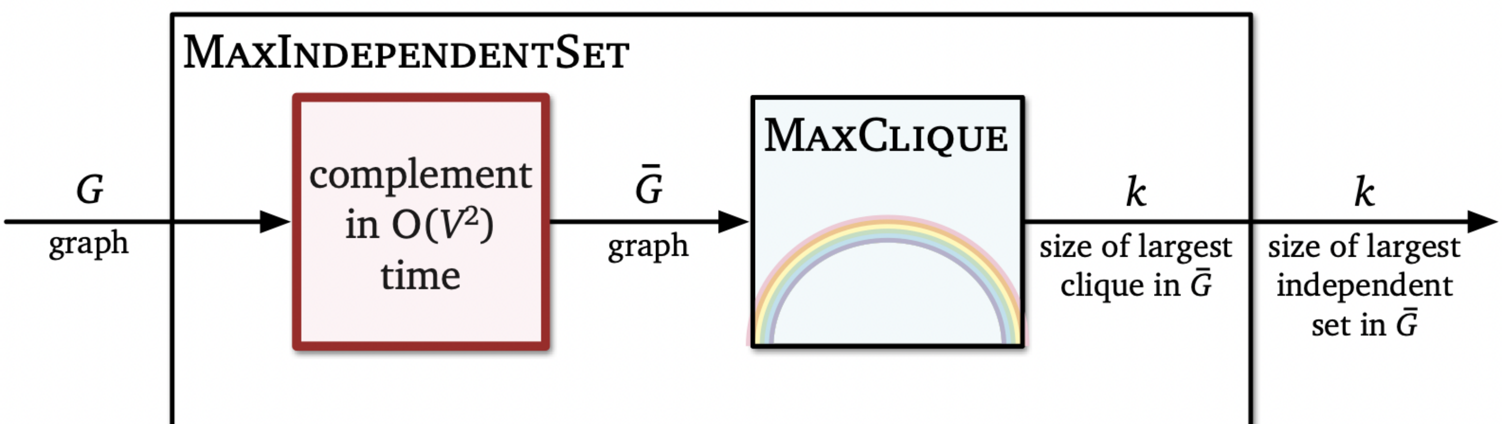
The **MaxClique** problem asks the number of nodes in its largest complete subgraph in a given graph.

A ***vertex cover*** of a graph is the set of vertices that touch every edge in the graph. The **MinVertexCover** problem asks the minimum number of nodes that touch every edge.



(fig: example of MaxIndSet, MaxClique, MinVertexCover).

MaxClique and MinVertexCover are both NP-hard. Plan:



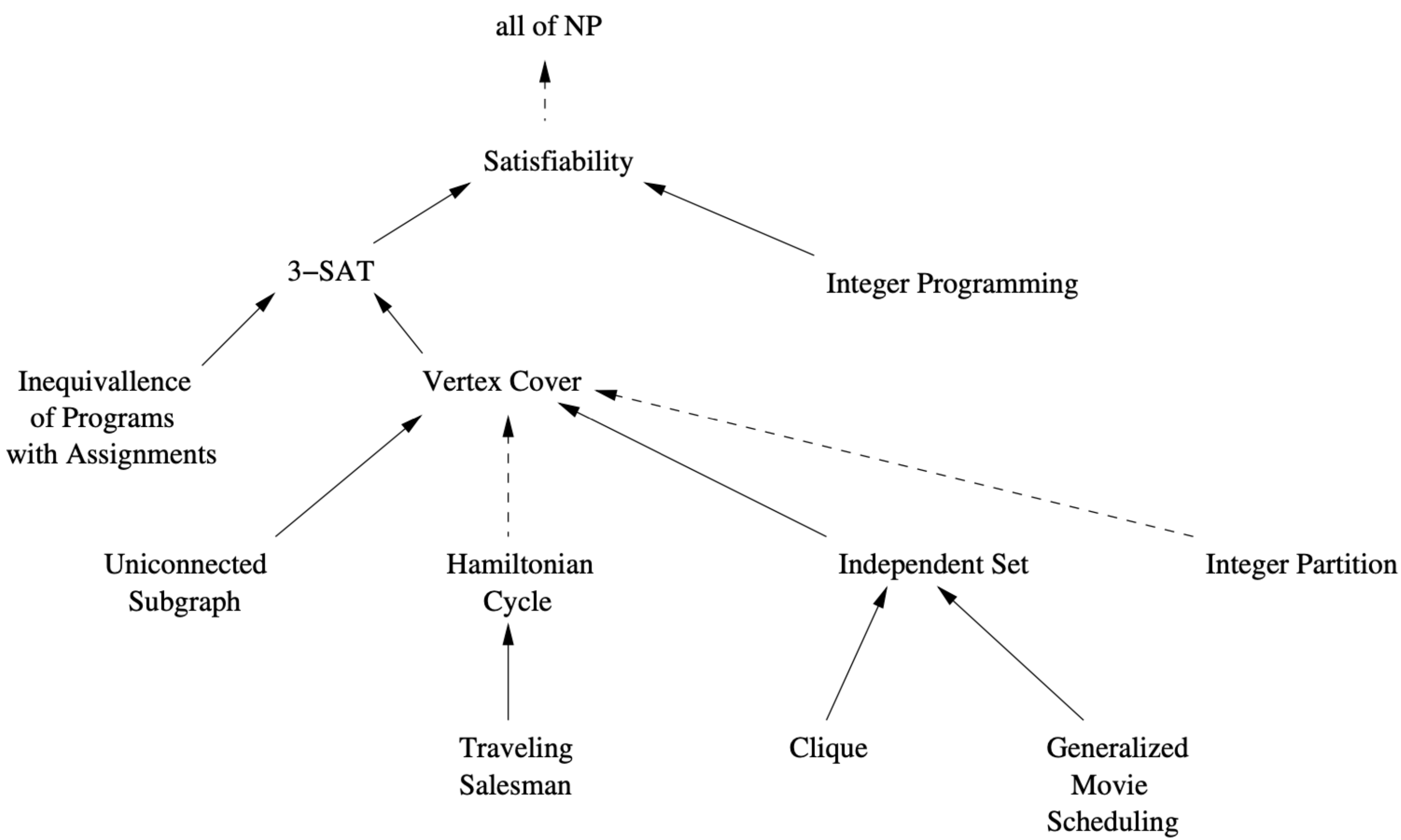
Edge complement of G: $\bar{G}$ has the same nodes as G, but complementary edge set.

- **MaxClique**: An independent set S in G is a clique in $\bar{G}$.

  S is MaxIndSet in $\bar{G}$ <=> S is MaxClique in G.

- **MinVertexCover**: Let S be an independent set in G. Then $V - S$ is a vertex cover. Therefore :

  S is MaxIndSet <=> V-S is MinVertexCover


all of NP
Satisfiability
3–SAT
Integer Programming
Inequivallence of Programs with Assignments
Vertex Cover
Uniconnected Subgraph
Hamiltonian Cycle
Independent Set
Integer Partition
Traveling Salesman
Clique
Generalized Movie Scheduling